Query-Flood DoS Attacks in Gnutella [*]

Neil Daswani and Hector Garcia-Molina
Computer Science Department
Stanford University
Stanford, CA 94305-9045
{daswani,hector}@db.stanford.edu

ABSTRACT

We describe a simple but effective traffic model that can be used to understand the effects of denial-of-service (DoS) attacks based on query floods in Gnutella networks. We run simulations based on the model to analyze how different choices of network topology and application level load balancing policies can minimize the effect of these types of DoS attacks. In addition, we also study how damage caused by query floods is distributed throughout the network, and how application-level policies can localize the damage.

Categories and Subject Descriptors

C.2.0 [**Computers-Communication Networks**]: General—*Security and protection*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems, Information networks*; I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Security, Algorithms, Measurement, Experimentation

Keywords

peer-to-peer, security, denial-of-service

1. INTRODUCTION

In this paper we study application-layer, flooding-based denial-of-service (DoS) attacks in peer-to-peer (P2P) systems, and propose simple techniques to cope with such attacks. Since there has been significant recent interest in DoS attacks, it is important to place our work in context.

First, we are addressing *application-layer* DoS attacks, as opposed to network-layer attacks. Most techniques that have been developed to date to deal with denial-of-service focus on network-layer attacks [3, 16, 33, 34, 22, 32, 7, 29, 8, 1, 19, 9, 18, 2, 38, 20]. Yet, public systems are also very vulnerable to application-layer attacks, even if they are immune to network-layer attacks. In a flooding-based application-layer attack, large numbers of application requests or messages (e.g., deliver email, open account, find page), or malicious application messages, can deny service to clients. These attacks can be more damaging, relative to the effort that the malicious party needs to expend, than network-layer attacks since each small message can cause a server to waste significant resources. Fortunately, since requests require the server to expend a significant amount of CPU and/or disk I/O resources, the server can afford to spend some time deciding which requests to honor. In contrast, routers processing network packets cannot afford to spend as much time making such decisions, else the overhead will be too high. Indeed, in this paper we propose some "policies" that may be too expensive for network-layer attacks, relatively speaking.

Second, the application we are considering is P2P systems, as opposed to other applications such as web search or e-commerce. For concreteness, it is important to focus on one particular type of system, and we chose Gnutella P2P systems (with supernodes) [30], because they are very vulnerable to attack and very popular. In Gnutella (as well as in Morpheus [17] and KaZAA [10]), a client submits a query (e.g., looking for a file) to a supernode (server). That node performs a local search, and also broadcasts the query to its neighboring nodes, asking them to also process the query. Thus, queries from a malicious client or server can exponentially multiply throughout the system, consuming resources that other clients cannot use. If we can manage DoS attacks in Gnutella, then we will surely be able to manage DoS attacks in other P2P systems that are less vulnerable than Gnutella, such as CAN [24], Chord [35], Tapestry [39], etc. And clearly, Gnutella systems are of

[*]This work was partially supported by NSF Grant CCR-0208683.

interest because they are the most prevalent P2P systems today, with over 25 million client downloads, and 300,000 concurrent users during peak periods.

Third, in this paper we advocate application-layer "load balancing" techniques that attempt to give clients a "fair share" of available resources, thus making it harder for malicious clients to deny service. Most DoS work to date does not fall in this category: current techniques tend to be either reactive, where in-progress attacks are detected, and services are denied to offending clients, or proactive, where security mechanisms prevent clients from gaining access to resources [11]. Instead, we do not require servers to distinguish attack queries from bonafide ones, and indeed, malicious clients will be able to receive some service. However, the load balancing policies try to make sure that offending clients do not receive an inordinate amount of service. Of course, in a Gnutella P2P system the challenge is to maintain a fair load distribution in spite of the multiplicative effect of query broadcast.

Clearly, load balancing policies do not eliminate the need for proactive and reactive techniques. We believe that all three types will be needed for protection against DoS attacks. In this paper, we simply focus on the load balancing techniques because they are also important and we feel they have not been studied adequately in this context. Because we are studying load balancing techniques, in our evaluations we focus on flooding-type DoS attacks, as opposed to attacks that are better dealt with by other techniques. (For example, if a single malicious query can crash a node, then we clearly need to ensure, using a proactive approach, that such a query is never executed.)

We also note that the load balancing techniques we advocate are not new. These types of techniques have been used for many years in network management, processor scheduling, and many other applications. Here we are simply applying these techniques to a P2P environment, and extending them to handle requests originating via flooding from a malicious node possibly multiple hops away.

However, one area where we have had to go beyond the current state of the art is in the *evaluation* of DoS load balancing techniques. In particular, we needed and developed techniques for modeling and quantifying the "usefulness" of load balancing DoS techniques. With our model and metrics we can compute how much "damage" a malicious node may cause, which network topologies may be more vulnerable to attacks (allowing greater damage), which nodes within a network are the most vulnerable, and which load balancing techniques may be most effective. We believe that such an evaluation is essential in order to get a handle on flooding-based DoS attacks.

Our main contributions in this paper are as follows:

- We define a simple but effective traffic model for query flow in Gnutella networks, and outline policies that nodes may use to manage query flow. We define expected behaviors for "good" and "malicious" nodes, and metrics that we use to evaulate the impact that malicious nodes have by flooding the network. (Sections 2, 3, and 4)
- We evaluate how network topology affects the ability of a malicious node to flood the network. In our evaluations, we study the vulnerability of complete, cycle, wheel, line, star, grid, and power-law topologies under various flow management policies. (Sections 5.1, 5.2, and 5.3)
- We evaulate how different combinations of flow management policies can be used to manage the distribution of damage across the nodes in a network. (Section 5.4)

2. GNUTELLA TRAFFIC MODEL

In this section we briefly describe a natural traffic model for a Gnutella P2P system that focuses on query flow and query load at nodes in the network. The model that we present is an intentionally coarse-grained and relatively simple model whose goal is to capture the important features of query traffic. We do not expect the model to predict actual query loads (as might be observed in a real network), but we do expect it to tell us about relative query loads at nodes of the network by using different application-layer policies.

The system is modeled as a graph with a set of supernodes V, and a connection topology, i.e., a set of point-to-point, bidirectional communication links, E. Our model divides time into a number of discrete time intervals, and explicitly models each supernode in the network. Supernodes typically have more processing power and disk I/O bandwidth than regular nodes. Supernodes are responsible for propagating queries throughout the network. Regular nodes connect to supernodes, and send their queries to supernodes to have them serviced. Regular nodes are not explicitly modeled.

Each supernode conducts three actions during each unit of time. At a given time t, each supernode 1) accepts and processes queries from adjacent supernodes, 2) accepts and processes queries it receives from regular nodes connected to it, and 3) forwards some combination of queries received from regular nodes and adjacent supernodes to adjacent supernodes. The time intervals are denoted by non-negative integers $t = 0, 1, 2, ..., K$. Our model can be used to approximate the continuous behavior of a P2P system as the physical time between intervals decreases.

Since our model mainly focuses on actions at the supernodes, the term node will be used to refer to a supernode in the remainder of the paper, unless we explicitly state otherwise. In addition, we will often refer to the regular nodes that are connected to a particular supernode j as j's *local peers*, and other supernodes that j is connected to as its *remote peers*. A *local query* is a query generated by a local peer, and a *remote query* is a query sent by a remote peer.

In this paper, we only model queries, and we do not model other messages (ping & pong, query-hit, and push), since query processing dominates the workload of a node.

Let $O_{j,k}(t)$ be the multi-set of queries that node j sends to node k at time t. If $(j,k) \notin E$, j does not have a connection to k and $O_{j,k}(t) = 0, \forall t$. Also, $O_{j,j}(t) = 0, \forall t$. We use multi-sets (bags) instead of sets because there may be duplicate queries (e.g., queries with the same search criteria).

Let $G_j(t)$ be the multi-set of queries that node j receives from local peers at time t. When queries are generated by local peers, they are assigned a time to live (TTL) that specifies the maximum number of nodes (regular or super) that the query may traverse in the network. Each node checks the TTL for each query, and does not forward any queries for which TTL=0. Nodes decrement the TTL for a query before forwarding it to other nodes. We assume that all peers in the network generate queries with the same TTL, τ.

Nodes have a limited processing capacity. As a result, a node j may not be able to process all of the queries it receives. At time t, j may have to choose some subset of the queries in the set $\bigcup_{i \epsilon V} O_{i,j}(t-1) \cup G_j(t-1)$ for processing.

Let $I_{i,j}(t)$ be the multi-set of queries that node j actually processes from node i at time t. ($I_{i,j}(t) \subseteq O_{i,j}(t-1)$.) If $(i,j) \notin E$, i does not have a connection to j and $I_{i,j}(t) = 0, \forall t$. Also, $I_{i,i}(t) = 0, \forall t$. Similarly, let $P_j(t)$ be the multi-set of local queries that j actually processes. ($P_j(t) \subseteq G_j(t-1)$.)

To illustrate, consider a system with two nodes, $V = \{1,2\}$ with topology $E = \{(1,2)\}$. Assume that $G_1(0) = \{q_1, q_2, q_3\}$. That is, node 1 receives 3 queries at time $t = 0$ from its local peers. At time $t = 1$, node 1 processes only two of the queries, so that $P_1(1) = \{q_1, q_2\}$. Assume that node 1 sends the two queries to node 2, so that $O_{1,2}(1) = \{q_1, q_2\}$. At time $t = 2$, node 2 receives $O_{1,2}(1)$, *but* it does not have to process all the queries. Depending on node 2's policy, it may decide to take say only the first query, so that $I_{1,2}(2) = \{q_1\}$.

A node j examines the incoming queries contained in $O_{i,j}(t-1)$, $\forall i$, as well as in $G_j(t-1)$. In our policies, we will consider the case in which node j may not have enough processing capacity (or does not care to) to examine all these queries (in which case it simply accepts them on a first-come-first-serve basis), as well as the case in which it has enough processing capacity to at least examine all these queries and then choose some subset of them for processing.

The time required to process a query may involve searching for a keyword in an inverted index, hash table, or some other main memory data structure. However, keyword search for content has its limitations, and over time more sophisticated search mechanisms will be employed. We have already started to see metadata search deployed in the LimeWire Gnutella client [13] and on FastTrack-based networks. As the search mechanisms become more complex, query processing will dominate the time required to service incoming queries. In turn, the time to examine all incoming queries will become negligible compared to the time required to process queries that are chosen to be serviced. Hence, while we examine both cases, we believe it will become important to make a good decision about which subset of the incoming queries to process.

During each time step, a node selects at most c_j queries for actual processing. A node never selects queries that it has seen before for processing. (This may occur due to cycles in the topology of length less than τ.) Thus, c_j represents the processing capacity of node j.

Our capacity constraint can be stated as follows: $\sum_{\forall i} |I_{i,j}(t)| + |P_j(t)| \leq c_j$.

Once a maximum of c_j queries have been accepted and processed, node j then broadcasts all of these queries to its adjacent nodes such that they too can determine if they have answers to the queries.

3. POLICIES

Given this framework, we are interested in understanding how nodes may manage DoS query traffic in a Gnutella network. There are a number of choices that each node has with regards to deciding what queries to accept and process.

If nodes make bad decisions, they may end up only accepting many "useless" queries that are generated by malicious nodes, wasting their processing capacity on these queries, and then forwarding these queries to adjacent nodes that will do the same. If a node makes good decisions, then it can minimize the effect of a flooding-based DoS attack. We assume that it is hard to distinguish a high load of legitimate queries from attack queries. As such, nodes must exercise some discretion in how to "fairly" allocate their processing capacity to servicing queries so as not to spend too much effort on queries that may be bogus.

In this section, we introduce some policies that nodes may use as options to manage query load. Once we have introduced these policies (and defined some metrics), we will present simulation results that determine which of these policies do best in minimizing the impact of malicious query floods in small networks.

3.1 Reservation Ratio (ρ)

Nodes must provide some level of fairness between servicing local and remote queries. If supernodes only service local queries, then local peers will not benefit from query results that could be obtained from other supernodes in the network. If supernodes service only remote queries, then local peers will be neglected.

To allow a supernode to decide how to split its processing capacity we define ρ to be the fixed fraction of query bandwidth that a supernode reserves to service local queries ($0 \leq \rho \leq 1$). A supernode j uses ρ to determine how many queries to accept from local peers and how many queries to accept from remote peers when the total number of queries sent to it exceeds c_j. (If the total number of queries sent to it does not exceed c_j, the supernode simply processes all the queries sent to it.)

More formally, if $|G_j(t-1)| \leq \rho c_j$, then $P_j(t) \leftarrow G_j(t-1)$. Otherwise, we select ρc_j queries from $G_j(t-1)$. Queries that are not selected from $G_j(t-1)$ for processing at time t can be queued for future time steps (or even discarded). The remaining capacity, $c_j - |P_j(t)|$ is now allocated among the queries arriving from remote peers. We will refer to ρc_j as node j's local query bandwidth (LQB), and $(1-\rho)c_j$ as node j's remote query bandwidth (RQB).

There are several combinations of policies that we shall consider for allocating the RQB amongst queries arriving from remote peers. There are two questions that these policies answer:

1) how many queries should be accepted from each remote peer?, and

2) if there are more queries arriving from a remote peer than we decide to accept, which ones should we accept?

The policy used to answer the first question is the *incoming allocation strategy*, and the policy used to answer the second question is the *drop strategy.*

3.2 Incoming Allocation Strategy

There are two key incoming allocation strategies (IASs) that we cover in this paper: Weighted and Fractional. We describe how each of these strategies select which adjacent nodes to process queries from in this section. In our descriptions, queries are represented as two-tuples $q=(o,t)$ where o is the node at which the query originated [1], and t is the query's current TTL. We refer to the components of the tuple using a dot notation such that $q.o$ refers to the origin node (at which the query was first generated) and $q.t$ refers to the TTL. Many distinct queries can have the same origin and TTL, and we will refer to η distinct queries with the same origin and TTL as ηq.

We will describe options for IASs and illustrate them using examples. In our examples, assume a graph with three nodes $V=\{1,2,3\}$ and $E=\{(1,2),(1,3)\}$. Also, for each of the nodes j, $c_j=100$, $1\le j\le 3$, and each node has $\rho=0.2$. In each of the examples, we present how node 1 decides how many and which queries to accept from nodes 2 and 3.

Weighted IAS. Weighted IAS is intended to model a "naive" Gnutella node in which the likelihood that a query from a particular incoming link will be accepted is proportional to the number of queries arriving on that link.

We assume that queries arriving at node j from remote peers are equally likely to be accepted. Thus, the more queries a neighbor sends, the more will get accepted.

If a total of less than $(1-\rho)c_j$ queries are sent by remote peers, then all queries that are sent are accepted for processing. If more than $(1-\rho)c_j$ queries are sent by remote peers, then the number of queries accepted from each remote peer is weighted by the fraction of the total queries sent. If a node has κ remote peers that send it $\alpha_1,\alpha_2,...\alpha_\kappa$ queries, it will accept up to $\frac{\alpha_\lambda}{\Sigma_{\forall i}\alpha_i}(1-\rho)c_j$ queries[2] from the λth remote peer, $1\le\lambda\le\kappa$.

For instance, let $|O_{2,1}(t-1)|=100$ and $|O_{3,1}(t-1)|=20$ with $(1-\rho)c_1$=80 and the LQB fully utilized. The Weighted IAS divides the RQB such that $|I_{2,1}(t)|=\frac{100}{120}(1-0.2)100=67$ and $|I_{3,1}(t)|=\frac{20}{120}(1-0.2)100=13$.

Fractional IAS. Fractional IAS is geared at giving each of a node's incoming links an equal fraction of query bandwidth. If a node has κ remote peers, a Fractional IAS allocates up to $1/\kappa$ of its query bandwidth for handling the queries from each of its remote peers. Any extra query bandwidth that is unused by a remote peer is allocated to remote peers. Also, if the LQB is not completely utilized by local peers, any leftover LQB is allocated to servicing queries from remote peers.

For example, let $|O_{2,1}(t-1)|=100$ and $|O_{3,1}(t-1)|=20$, and node 1's RQB is 80, as before. Assume also, as before, that the LQB is completely utilized. If node 1 uses a Fractional IAS, $\frac{(1-\rho)c_1}{2}=\frac{(1-0.2)100}{2}$=40 queries are allocated to each remote peer, but the extra 20 queries per unit time that are not used by node 3 are allocated to node 2. As a result, $|I_{2,1}(t)|=40+(40-|O_{3,1}(t-1)|)=40+20=60$ and $|I_{3,1}(t)|=20$.

Other policies. There exist many other possible IAS policies (i.e., least queries first, preferred neighbors first, etc.). For the remainder of this paper, we only consider Fractional and Weighted IASs. However, other IASs may warrant examination in future study.

3.3 Drop Strategy (DS)

This section describes drop strategies (DSs). When the IAS used for node j determines that no more than m queries may be accepted from a remote peer i, and i sends $|O_{i,j}(t-1)|=m+\Delta$ queries (where $\Delta>0$), node j uses a DS to determine specifically which Δ queries to drop. In our examples below, node j receives $O_{i,j}(t-1)=\{2q_1,2q_2,6q_3\}$ where $q_1=(a,5)$, $q_2=(a,4)$, and $q_3=(b,4)$.

The Proportional and Equal strategies described below make decisions about which queries to drop by considering the nodes at which the queries in $O_{i,j}(t-1)$ originated as well as their TTL.

Proportional. Let node j receive $O_{i,j}(t-1)=\{\ \eta_1q_1, \eta_2q_2,\ ...\ \eta_nq_n\}$. If j uses a Proportional DS, it will accept up to $\frac{\eta_\chi}{\Sigma_{\chi=1}^{n}\eta_\chi}$ queries of type q_χ, $1\le\chi\le n$.

In our example, if $m=5$, then Proportional DS chooses $I_{i,j}(t)=\{q_1,q_2,3q_3\}$.

Equal. The Equal DS chooses queries uniformly based on the origin of the query. If queries arrive at j from β different sources (not necessarily neighboring nodes), the Equal DS will attempt to choose $\frac{m}{\beta}$ queries from each source. If some sources sent fewer than $\frac{m}{\beta}$ queries, then the extra query bandwidth will be shared equally across queries from sources that sent more than $\frac{m}{\beta}$ queries.

For instance, if $m=3$, then the Equal DS chooses $I_{i,j}(t)=\{q_1,q_2,q_3\}$.

[1]Current Gnutella networks do not stamp queries with the nodes at which they originated, but this feature could be added to support load balancing.

[2]To keep our explanation of policies conceptually clear, we will not add floors and ceilings to quantities. In our simulations, floors are taken for most calculated quantites, and policies are run in multiple "rounds" to ensure that all available query bandwidth is used up.

PreferHighTTL / PreferLowTTL. These strategies are used to drop either those queries with the lowest or highest TTLs, regardless of the nodes at which they originated.

If $m = 1$, PreferLowTTL gives either $\{q_2\}$ or $\{q_3\}$. (Ties are broken arbitrarily.) Alternatively, for $m = 1$, PreferHighTTL gives $\{q_1\}$.

4. METRICS

To evaluate whether or not (and how well or how badly) the policies above may help us manage queries distributed by malicious nodes, we define a work metric, the concept of a service guarantee, and a damage metric that allow us to quantitatively determine the service loss a malicious node may be able to inflict on a network. In addition, we will describe how we model "good" nodes (that use our policies) and how we model "malicious" nodes (that attempt to flood the network, possibly ignoring reasonable policies).

4.1 Work

Our definitons for work broadly measure the number of queries processed by one or more nodes in the network. More specifically, for a particular node j, $W_j(t)$ is the *work* or cumulative number of queries processed at node j from time 0 to time t. Furthermore, we distinguish *local work* from *remote work*. The local work, $L_j(t)$ is the cumulative number of queries that node j receives from its local peers and processes from time 0 to time t. Similarly, remote work, $R_j(t)$ is the cumulative number of queries that node j receives and processes from its remote peers from time 0 to time t. Of course, $W_j(t) = L_j(t) + R_j(t)$.

To understand how local and remote work changes with ρ, let us consider what happens if we start with $\rho = 0$ and slowly increase it.

If $\rho = 0$ for all nodes, then each of the nodes allocates all of its query bandwidth to queries arriving from remote peers. Unfortunately, nodes send out $\rho c_j = 0$ queries during each time step. While each node is "all ears," no node is sending out any work. As a result, the total local work and total remote work are both 0.

As ρ increases, more and more queries are accepted from local peers, and more and more queries will be processed by the network. Both the local and remote work will increase. However, at some point, each node will be processing the maximum number of queries possible (as specified by its capacity, c_j). After this point, if ρ increases any further, nodes will have to start dropping remote queries, and the amount of remote work will start decreasing. However, since we make the assumption that local peers always generate ρc_j queries, the amount of local work will continue increasing as ρ increases.

Once $\rho = 1$, then each of the nodes allocates all of its query bandwidth to queries arriving from local peers, and do not service any queries from each other. The total local work will be maximum and the total remote work will be 0.

While the query bandwidth of each of the nodes may be fully utilized when $\rho = 1$, users do not receive the benefit of having their queries processed at other nodes. To maximize the number of queries processed at remote nodes, we can set ρ to maximize the remote work. In the following section, we show how to set ρ to do this.

4.2 "Good" nodes

In our model, "good" nodes have two important characteristics. Firstly, we make the simplifying assumption that the processing capacity c_j is the same for all nodes in the graph. In particular, $\forall j \epsilon V, c_j = C$, where C is some constant. Secondly, good nodes are compelled to find a setting for ρ that maximizes the remote work [3].

DEFINITION 4.1. Optimal Rho, $\hat{\rho}$. *Let $\hat{\rho}$ be the setting for ρ that maximizes $\Sigma_{j \epsilon V} R_j(t)$.*

We may analytically solve for $\hat{\rho}$ for simple network topologies, as we will demonstrate shortly, and we may approximate or experimentally determine $\hat{\rho}$ for more complex topologies.

Also, it is the case that for certain topologies the optimal value for ρ may be different for different nodes in the network. However, for simplicity, we will assume that we would like to have a common setting for ρ for all nodes. We will have to sacrifice some remote work to have a common ρ, but doing so will simplify the implementation of our load balancing policies in a real network.

Consider the network topology $K_3 = (\ V = \{1, 2, 3\}$, $E = \{(1, 2),(1, 3),(2, 3)\})$ in which we have a network of three nodes with three edges completely connecting the nodes, $c_j = 100, 1 \leq j \leq 3$, and $\tau = 1$. For K_3, the reader can verify that the setting at which ρ maximizes the remote work is $\frac{1}{3}$. (At this setting, the amount of new work generated and sent to any given node is exactly equal to the amount of work that it can accept.)

While the appropriate setting for ρ might be obvious in our small example, it is important for good nodes in our network to be able to compute or approximate $\hat{\rho}$ for arbitrary networks. We provide a formula for computing $\hat{\rho}$ for "symmetric" networks (such as complete, cycle, or hypercube networks) below, following some elementary definitions.

DEFINITION 4.2. Distance, $d(j, k)$. *Let $d(j, k)$ be the length of the shortest path between nodes j and k. Note that $d(j, j) = 0$.*

DEFINITION 4.3. Radial Node Set, $\delta(j, h)$. *Let $\delta(j, h) = \{\ v \mid d(j, v) = h\ \}$. That is, $\delta(j, h)$ is the set of nodes v such that the shortest distance between j and v is exactly h.*

DEFINITION 4.4. Arial Node Set, $D(j, h)$. *Let $D(j, h) = \bigcup_{i=1}^{h} \delta(j, i)$. That is, $D(j, h)$ is the set of nodes v such that the distance between j and v is greater than or equal to 1 but less than or equal to h. Note that $j \notin D(j, h)$. Informally, $D(j, h)$ is the set of nodes that are within h hops of j, not including j itself.*

[3]Alternatively, we can maximize the total work, but maximizing the remote work has the benefit that it gives us the smallest possible setting for ρ for which the total work is maximized and the minimum number of remote queries are dropped. A more detailed discussion appears in [23].

THEOREM 4.1. Optimal Rho ($\hat{\rho}$) for Symmetric Networks. *Suppose that for all nodes $j \in V$ have $c_j = C$ for some constant $C > 0$, $|D(j, \tau)| = D$ for some constant $D > 0$, and all nodes have ρ set to the same value, then $\hat{\rho} = 1/(D + 1)$.*

We provide the proof of this theorem in Appendix A.

For more complex networks, such as those studied in our evaluations in Section 5, we experimentally determined $\hat{\rho}$. In future work, we plan to study how to calculate good approximations for $\hat{\rho}$ for arbitrary networks.

In summary, good nodes in our model set $c_j = C$, and $\rho = \hat{\rho}$ to maximize the remote work done by the network.

4.3 Malicious Nodes

We are interested in studying flooding-based attacks, and we model a malicious node such that it generates as many queries as it is capable of. However, there exist many other behaviors that a malicious node may engage in to cause harm to other nodes in the network. While there are many such options available to the adversary, we focus specifically on query floods in this paper.

To construct a query flood attack, a malicious node dedicates all of its processing capacity to generating "useless" queries. Malicious nodes may be able to generate more than C queries. However, since a good node knows that other good nodes can send at most C queries, it only examines the first C queries from each incoming link during a given time step, and ignores the rest. While a malicious node can generate more than C queries, the effect will be the same as if it generates exactly C queries. Hence, we set $c_m = C$, where m is a malicious node.

After generating queries in a given time step, a malicious node has no processing capacity left over. In addition, it does not have any incentive to process or forward queries that are sent to it by remote peers. To model a flood generated by a malicious node, we have the malicious node set ρ to 1, whereas good nodes typically set ρ to a significantly lower value.

4.4 Service

A key metric that we can use to understand the effects of a malicious node in the network will be "service." The *service*, $S_{i,j}(t)$, is the number of queries that originate at node i and are processed at node j at time t. The service $S_{i,j}(t)$ tells node i how many of its queries are processed by node j at time t. For example, if node 2 processes 5 of node 1's queries at time $t = 3$, then $S_{1,2}(3) = 5$.

We now more formally define the notion of service, and two variations of it, radial and arial service, that we use in our evaluations.

DEFINITION 4.5. Service, $S_{i,j}(t)$. *Let*

$$S_{i,j}(t) = \sigma_{q.o=i}(\bigcup_{v \epsilon V} I_{v,j}(t))$$

Note that we use σ to be selection over multi-sets, as defined in bag-relational algebra.

DEFINITION 4.6. Radial Service, $R_j(h, t)$. *Let $R_j(h, t) = \Sigma_{v \epsilon \delta(j,h)} S_{j,v}(t)$. $R_j(h, t)$ denotes the total service that node j receives from all of the nodes whose shortest distance from j is exactly h. (Informally, $R_j(h, t)$ is the total service that j receives from all nodes that are exactly h hops away from j in the network.)*

DEFINITION 4.7. Arial Service, $S_j(h, t)$. *Let $S_j(h, t) = \Sigma_{v \epsilon D(j,h)} S_{j,v}(t)$. $S_j(h, t)$ denotes the total service that node j receives from all of the nodes within h hops.*

4.5 Worst-case Scenario

In the evaluations described in Section 5, we consider a "worst-case" scenario in which there is a single malicious node in a small network of "fully-loaded" nodes.

In our "worst-case" scenario, we assume that all good nodes in the network are broadcasting $\hat{\rho}C$ queries. In other words, $\forall j, t \; G_j(t) = \infty$ where j is a good node, and the reservation ratio that the good nodes choose is $\rho = \hat{\rho}$. The malicious node, as before, has its reservation ratio set to $\rho = 1$. The worst-case might be said to model a real network at 4pm in the afternoon when it is at its peak load.

4.6 Victim Nodes

In some of our evaluations, we will study the effect of the malicious node on the network from the point of view of a "victim" node. In particular, we will be interested in understanding what is the reduction in service that the victim node receives if there is a malicious node present in the network.

As mentioned above, a malicious node in our evaluation is one that sets $\rho = 1$ in an attempt to flood the network with "useless" work. The malicious node does not carry out any behavior that explicitly attacks the node that we will call the victim. Nevertheless, we will still use the term victim for the node from whose point of view we are studying the impact of the malicious node's behavior. More specifically, we study the degradation in service that the victim node experiences due to the presence of the malicious node. It is most likely the case that other nodes suffer degradation in the service they receive from other nodes as well. However, by studying service degradations for different victim nodes in the network, we build an understanding of how much impact a malicious node has on various relative placements of the malicious and victim nodes in a particular topology.

We define the service that the victim node receives from the network in a worst-case scenario as a *service guarantee*.

DEFINITION 4.8. Service Guarantee, $S_j(t)$. *Let $S_j(t) = S_j(\tau, t)$. $S_j(t)$ denotes the total service that node j receives from all of the nodes within τ (TTL) hops.*

4.7 Damage

With all of this machinery in place, we are now ready to quantify the degradation in service that might be brought about by a malicious node. We call this degradation in service *damage.*

In the following definitions, $S_j(t)$ refers to the service guarantee that j receives from the network when $\underline{\text{there}}$ is no malicious node present in the network, and $\overline{S_j(t)}$ refers to the same quantity when there does exist a malicious node in the network.

Damage with respect to a victim node j, $D_j(t)$, is defined as follows:

DEFINITION 4.9. Damage for Victim Node j

$$D_j(t) = \frac{S_j(t) - \overline{S_j(t)}}{S_j(t)}$$

If $S_j(t) = \overline{S_j(t)}$, then the malicious node is not able to affect the service guarantee that j receives from the network at time t, and the corresponding damage is 0. On the other hand, if $\overline{S_j(t)} = 0$, then the malicious node is able to prevent j from receiving any service at all at time t, and the corresponding damage is 1.

We define cumulative network damage as the sum of the loss in service incurred by every node in the network from time 0 to time t.

DEFINITION 4.10. Cumulative Network Damage

$$D(t) = \frac{\Sigma_{i=0}^{t} \Sigma_{j \epsilon V} (S_j(i) - \overline{S_j(i)})}{\Sigma_{i=0}^{t} \Sigma_{j \epsilon V} S_j(i)}$$

Similarly, the damage is 0 if the malicious node is not able to have an effect on the network, while the damage is 1 is the malicious node is able to prevent all remote work from taking place in the network.

Finally, we define cumulative radial damage as the reduction in service that a node j experiences at nodes h hops away due to the presence of a malicious node.

DEFINITION 4.11. Cumulative Radial Damage. $D_j(h,t)$ $= \frac{\Sigma_{i=0}^{t} \Sigma_{j \epsilon \delta(j,h)} (R_j(h,i) - \overline{R_j(h,i)})}{\Sigma_{i=0}^{t} \Sigma_{j \epsilon \delta(j,h)} R_j(h,i)}$. *$D_j(h,t)$ denotes the damage, or reduction in service that node j receives from all of the nodes whose shortest distance from j is exactly h.*

5. RESULTS

In this section, we present the results of evaluations run using a simulator that we developed called Fargo.[4] Fargo implements the traffic model described in Section 2, allows us to choose any of the policies described in Section 3 for a given network topology, and measures the metrics defined in Section 4.

We chose to evaluate small network topologies and a single malicious node to build a fundamental understanding of the issues and trade-offs that a system architect would need to keep in mind when designing a flood-tolerant system.

All of our evaluations were run on small networks of either 14 nodes (for complete, cycle, wheel, line, and star topologies) or 16 nodes (for grid and power-law topologies[5]) with a single malicious node in the graph, and all queries were constructed with a TTL (τ) of 7. In the simulations, each node is given a maximum processing capacity of $C = 10000$ queries per time step, and $\hat{\rho}$ was experimentally determined to within 0.01 of the actual value. Each of the evaluations was run for $t = 100$ time steps which was sufficient to attain steady state in all cases.

Table 1 shows the cumulative damage incurred for different network topologies for the strategies outlined in Section 3. The results shown in this table assumed a worst-case scenario as defined in Section 4.

The first column of the table lists the topology used for a particular simulation. For topologies for which it made sense, simulations were done in which the malicious node was placed in different positions in the network, and the position of the malicious node is indicated in parenthesis.

For example, for a star topology, the malicious node could either be in the center of the star, or at one of the spokes. As might be expected, when the malicious node is in the center of a star topology, nodes at the spokes are unable to answer each other's queries at all and the resulting damage is 1.

The results in Table 1 help us answer the following questions in the indicated sections:

- Which IAS and DS strategies minimize damage the best? (and which strategies are the worst at minimizing damage?) Does the best IAS / DS strategy depend upon the topology? Or do different IAS/DS strategies work better for different topologies? (Section 5.1)
- For a given topology, how much can the damage be minimized by using the best IAS/DS compared to other strategies? (Section 5.2)
- For a fixed IAS/DS strategy, how does topology affect damage? Are there certain topologies that are less prone to damage than others? Are some nodes particularly susceptible to attack? (Section 5.3)
- How is damage distributed across the network? How do different combinations of policies affect the distribution of damage? (Section 5.4)

5.1 IAS/DS Policies and Damage

Fractional IAS can be used with Equal or PreferHighTTL DSs to minimize damage independent of network topology. Weighted IAS and PreferLowTTL DS maximize damage independent of network topology.

From Table 1, we can see that the combination of the Fractional IAS together with either the Equal or PreferHighTTL DSs minimize damage independent of topology and the location of the malicious node. The Fractional IAS limits the maximum number of queries that arrive from a particular link in the face of an overabundance of queries. All nodes that are adjacent to a malicious node will accept only some fraction of the malicious node's queries, and all nodes that are two hops away from the malicious node will only accept some fraction of that fraction. As such, the number of malicious queries that are received by a node drops off quickly with the node's distance away from the malicious node. Of those queries that are received from adjacent nodes, the Equal DS fairly distributes available query band-

[4]Our simulator is named after a city in North Dakota that is frequently flooded by the Red River of the North that runs through it. More information about Fargo, North Dakota and the Red River is available at http://www.ndsu.nodak.edu/fargoflood.

[5]For the results shown in this paper, we used a particular instance of a power-law topology as described in [23]. We are in the process of extending our simulator to randomly generate a number of power-law topologies, and average the results over the generated topologies.

	Fractional				*Weighted*			
Topology (Location)	*Prop*	*Equal*	*PfHighTTL*	*PfLowTTL*	*Prop*	*Equal*	*PfHighTTL*	*PfLowTTL*
Complete	0.143	0.143	0.143	0.143	0.545	0.545	0.545	0.545
Cycle	0.388	0.314	0.312	0.533	0.527	0.459	0.387	0.695
Grid (Center)	0.273	0.227	0.274	0.292	0.454	0.363	0.422	0.569
Grid (Corner)	0.225	0.170	0.187	0.286	0.371	0.270	0.247	0.570
Grid (Edge)	0.282	0.191	0.208	0.378	0.412	0.306	0.294	0.553
Line (Center)	0.324	0.248	0.330	0.515	0.428	0.306	0.398	0.609
Line (End)	0.175	0.148	0.143	0.275	0.219	0.184	0.165	0.346
Power-Law (High)	0.272	0.262	0.284	0.324	0.539	0.505	0.484	0.612
Power-Law (Low)	0.201	0.169	0.193	0.267	0.443	0.367	0.386	0.534
Star (Center)	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Star (Edge)	0.142	0.143	0.142	0.143	0.526	0.506	0.542	0.545
Wheel (Center)	0.386	0.386	0.386	0.386	0.726	0.751	0.717	0.751
Wheel (Edge)	0.335	0.337	0.354	0.388	0.505	0.444	0.510	0.573

Table 1: Total Cumulative Network Damage as a function of topology, IAS, and DS

width based on the origin of the queries, so the malicious node's queries are given the same weight as queries from other nodes, even if the malicious nodes sends many, many more of them. The PreferHighTTL DS performs well also because malicious queries stay localized– they are given lower and lower preference as they travel away from the malicious node.

The Equal and PreferHighTTL DSs perform comparably in many cases, although the Equal strategy performs better in general in most topologies we considered. In the few cases that PreferHighTTL performed better, the difference was marginal. However, Equal DS requires queries to be "stamped" with the node at which they originated, whereas PreferHighTTL does not require this extra information. Currently deployed Gnutella networks do not stamp queries, and the PreferHighTTL DS is a good substitute as it achieves comparable performance without stamping.

Weighted IAS always incurs more damage independent of DS and topology. We can expect that most Gnutella nodes deployed today can be modeled as using a Weighted IAS as they arbitrarily accept queries from other nodes without enforcing any traffic management policy. The Weighted IAS allows queries that are part of a flood to have a significantly higher chance of being chosen for processing relative to legitimate queries.

In general, when nodes use the Weighted IAS, damage increases as the average connectivity of the nodes increases. On the other hand, when nodes use the Fractional IAS, damage decreases as the average connectivity of the nodes increases.

The PreferLowTTL DS never reduces damage, and often results in significantly more damage as compared to the other DSs. One potential reason to use PreferLowTTL might be to attempt to increase the "reach" of a query, and to attempt to obtain as many search results from nodes a great distance (but less than TTL hops) away from the originator of the query. Unfortunately, when the malicious node is the originator of a query, PreferLowTTL allows its queries to be spread as far as possible and incur a large amount of damage.

5.2 Damage Reduction

Damage reductions of 1.4 to 3.8 times can be achieved with Fractional/Equal IAS/DS, depending upon topology (see Table 2).

Table 2 shows the damage reduction factors that can be achieved by switching from a Weighted/Proportial IAS/DS to a Fractional/Equal IAS/DS for all of the topologies considered with the malicious node in the most threatening position. For example, employing Fractional/Equal IAS/DS for the power-law topology results in reducing damage by about a factor of two as compared to Weighted/Proportional IAS/DS when the malicious node is highly connected.

To put this damage reduction factor in perspective, it is worthwhile to remember that we measure damage in a worst-case scenario, when the network is "fully-loaded" as defined in Section 4.5. At a time at which the network is *not* heavily loaded (and has no malicious node), a node is able to have many of its queries serviced; the number of queries that it has serviced by other nodes is greater than its service guarantee. When the network is at its busiest (4pm in the afternoon on a weekday), again with no malicious node, a node receives an amount of service that is exactly equal to its service guarantee. A node might have, for instance, 200 of its queries processed at other nodes. Our damage metric (as shown in Table 1) tells us how many queries a malicious node is able to rob the good node of at this busiest time. If the damage is 0.5, then the malicious node is able to rob the good node of 100 queries. By using a better IAS and DS policy, we might be able to reduce the damage. If the new damage using the better policies is 0.25, then we are able to recover 50 queries for the victim node; that is, other nodes will service 50 additional queries for the victim by using better policies when the malicious node is present. The damage reduction factor in this case is $\frac{0.5}{0.25} = 2$.

The damage reduction factors for various topologies and policies are shown in Table 2.

Topology (Location)	*F/E*	*W/P*	*Dmg Red Ftr*
Complete	0.143	0.545	3.8
Cycle	0.314	0.527	1.7
Grid (Center)	0.227	0.454	2.0
Line (Center)	0.248	0.428	1.7
Power-Law (High)	0.262	0.539	2.1
Star (Center)	1.000	1.000	1.0
Wheel (Center)	0.386	0.726	1.9

Table 2: Damage Reduction Factor using Frac/Equal IAS/DS

5.3 Damage vs. Topology

The complete topology under the Fractional IAS is the least prone to damage, and is insensitive to the position of the malicious node.

Figure 1 shows how damage varies with topology and placement of the malicious node. Figure 1 graphically depicts the Weighted/Proportional and Fractional/Equal columns of Table 1. The results corresponding to the star topology with the malicious node in the center have been excluded as the damage is always 1, and the exclusion allows the reader to see other the results with better resolution. Also, the names of the graph topologies have been abbreviated (K=Complete, C=Cycle, W=Wheel, L=Line, S=Star, P=Power-Law, G=Grid).

From Figure 1, we can see that if the malicious node can take on any position in the network, then the complete topology minimizes damage. Of course, the use of Fractional IAS plays a significant role in the complete topology's ability to minimize damage. The more links that a node using Fractional IAS has, the less negative of an impact can be caused by a single malicious node connected to it. In Table 1, it is interesting to note that due to the symmetry of the actions taking place at each node, all of the drop strategies that we consider perform equivalently in a complete network.

From Table 1, we also learn that topology alone cannot significantly reduce damage if bad policies are used. If a Weighted/PreferLowTTL IAS/DS is used, a malicious node can cause a damage of at least 0.5 for all topologies in the most threatening position. By contrast, if Fractional/Equal IAS/DS is used, then the worst possible damage is 0.386. Hence, it is important to use good policies regardless of the topology of the network.

In all topologies, we find that damage increases as the connectivity of the malicious node increases. In addition, we find that the closer the malicious node is to the "center" of the network, the more damage it can cause. Therefore, when new "untrusted" nodes join a network, they should be confined to the "edges" of the network. Over time, nodes that persist in offering service can be moved towards the center. In today's Gnutella networks, nodes can join at any random location and no explicit mechanism exists to incrementally move nodes towards the center of the network based on a node's "history."

Of course, a malicious node can "act" good until it finds itself in a central position in the network, and can start flooding at that time. Hence, while good policies can minimize the damage, it will be important to develop techniques that can detect and disconnect malicious nodes. Since good nodes in our model should generate no more than $\hat{\rho} c_j$ new queries per time step when there is high load, it might be worthwhile to disconnect any node that is sending more than $\hat{\rho} c_j$ queries under a high load condition. However, in a real network, malicious nodes can easily forge source IP addresses, and can make it appear as if they are "good nodes" that are just forwarding queries that were generated elsewhere. Nevertheless, while the idea of moving "trusted" nodes to the center does not prevent bad nodes from masquerading as good ones, it does "raise the bar" for an attacker to move into a more threatening position.

5.4 Damage Distribution

Fractional/Equal IAS/DS minimizes flood damage distributed in a cycle topology.

In this section, we measure how damage due to a single malicious node is distributed across the network. Due to space limitations, we only discuss damage distribution for the cycle topology here.

Damage distribution is measured with respect to "victim" nodes in the network. We examine the relation between IAS/DS policies and damage distribution.

Figure 2 shows the damage incurred at the victim node when the victim and malicious nodes are separated by 1, 3, 5, and 7 hops in a cycle topology. In general, damage decreases as the distance between the victim and malicious node increases.

Since damage decreases as distance from a malicious node increases, good nodes should attempt to make new connections in a way that distances them from malicious nodes. One method by which nodes can attempt to distance themselves from malicious nodes is by connecting to nodes that they "trust." That is, if a node i is reasonably sure that another node j is not malicious, then i should connect to j. Node j may be run by a friend of node i, or, in an enterprise setting, node j may have a business relationship with i. In either case, if node i connects to a "trusted" node j, then i can be reasonably sure that it has inserted at least one hop between itself and some malicious node that is part of the topology. Node j benefits from the same.

In the case that a node does not have any "friends," but can use a Fractional IAS, it should make many connections to shield itself from a potential flooding attack. If it makes m connections, then it accepts a maximum of $\frac{1}{m}$ useless queries from a malicious node. However, if a "friend-less" node is only capable of using a Weighted IAS, then it should connect to just a few nodes. The more nodes that it connects to, the higher the probability that it will connect to a malicious node.

In addition, nodes should attempt to connect to other nodes that are either themselves highly connected and using a Fractional IAS, or lowly connected and using a Weighted IAS. The less "exposed" that a node's neighbors are to flooding, the less exposed the node itself will be to flooding.

Figures 3 and 4 show how the damage incurred by the victim node in Figure 2 is distributed from 1 to τ hops away. Lines are plotted for different configurations

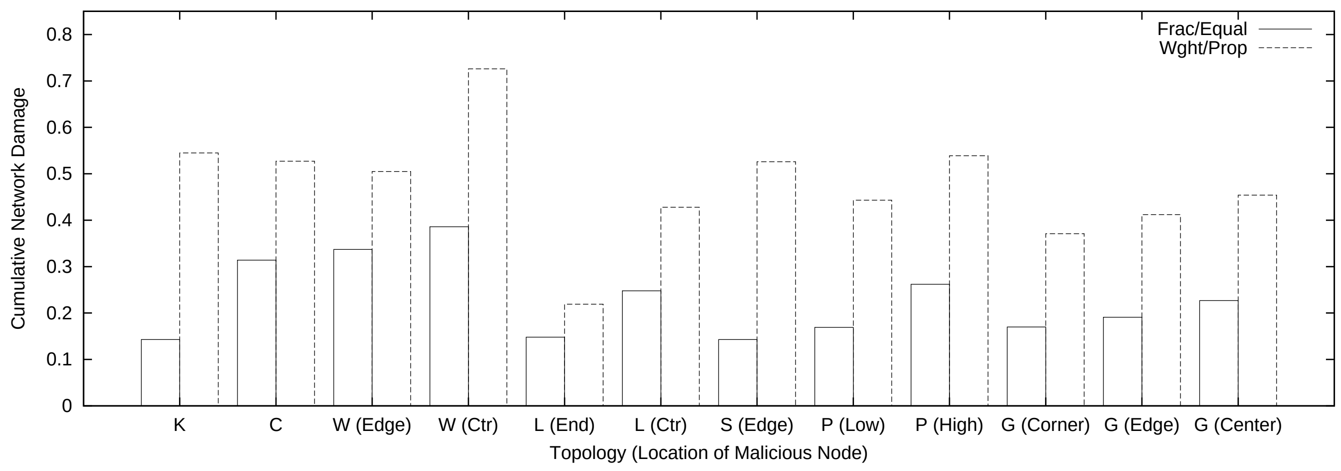

Figure 1: Damage vs. Topology for Fractional/Equal and Weighted/Proportional IAS/DS

of the victim and malicious node in which the distance (d) between them are 1, 3, 5, and 7 hops. Each (x,y) point on a line shows the reduction in service (y) that the victim receives x hops away.

In our following discussion, we will use the terms "upstream" and "downstream." Upstream refers to the direction closer to the malicious node, and downstream refers to the direction farther away from the malicious node.

Similar to the bar chart in Figure 2, Figures 3 and 4 show that more damage is incurred at upstream nodes that are closer to the malicious node (and further from the victim).

However, they also show how the damage is distributed at various distances away from the victim node. The damage (y) in Figure 3 incurred by the victim is averaged over both nodes that are x hops away from the victim in the cycle. When the distance (d) between the victim and the malicious node is 1, the damage is always greater than the damage when the distance between them is 7, as can be seen by the fact that the $d = 1$ line is always higher than the $d = 7$ line.

What we could not see in Figure 2 is that the extent of the damage one hop away from the victim is much more significant when the malicious node is one hop away than when the malicious node is seven hops away. When the malicious node is one hop away, the victim can only receive service from the good node that is one hop away, so the damage is at least 0.5. In addition, the victim is forwarding the flood queries from the malicious node to the good node that is one hop away. In Figure 3, the good node is using the Proportional DS, and, as a result, drops some of the victim's queries while attempting to process the flood of queries that arrives. The damage one hop away is therefore more than 0.5 in the $d = 1$ case; it is 0.61 in Figure 3. The damage two hops away is even more (0.68). Firstly, the victim's queries are never able to reach the upstream node two hops away because the malicious node never forwards them (contributing 0.5 to the damage). Secondly, since the downstream node one hop away does not accept all of the victim's queries, it does not forward all of the victim's queries to the downstream node two hops away. Of those queries that are forwarded, the downstream node two hops away accepts only a proportion of the victim's queries (incuring an additional 0.18 damage).

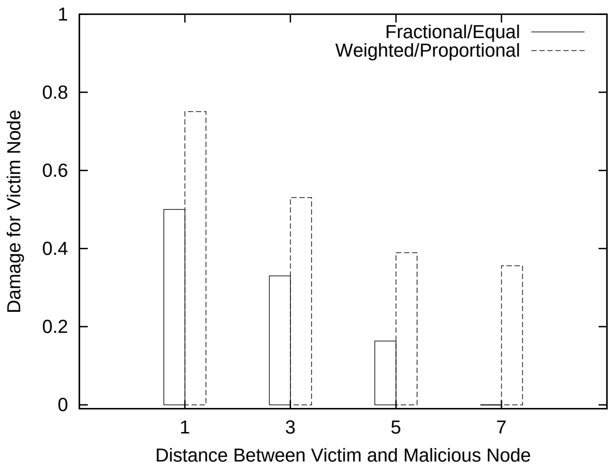

Figure 2: Damage vs. Distance from Malicious Node in a Cycle Topology

In summary, there are two types of damage that are caused by the malicious node. *Structural damage* is caused because a malicious node does not process or forward queries itself. When the malicious node is one hop away from the victim, the structural damage is 0.5 one hop away since the malicious node does not process any of the victim's queries. A second type of damage, *flood damage*, is caused by the traffic that the malicious node creates. When the malicious node is one hop away, and we are using a Proportional DS, there is flood damage that occurs at the good node that is one hop away. Due to the malicious query traffic that is forwarded to the good node, the good node cannot process all of the victim's queries. The flood damage in this case is 0.11.

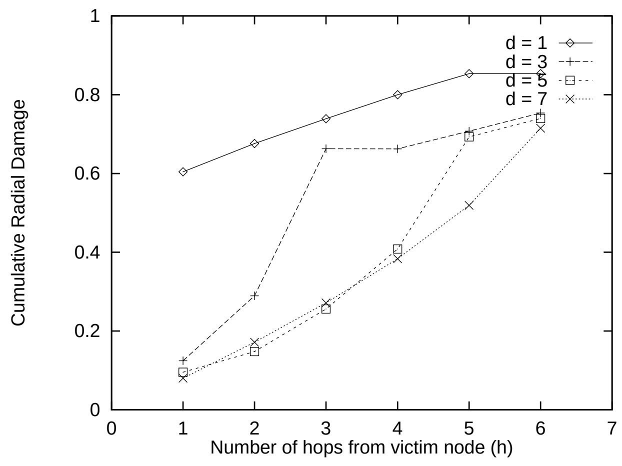

Figure 3: Damage Distribution for a Cycle with Weighted/Proportional IAS/DS

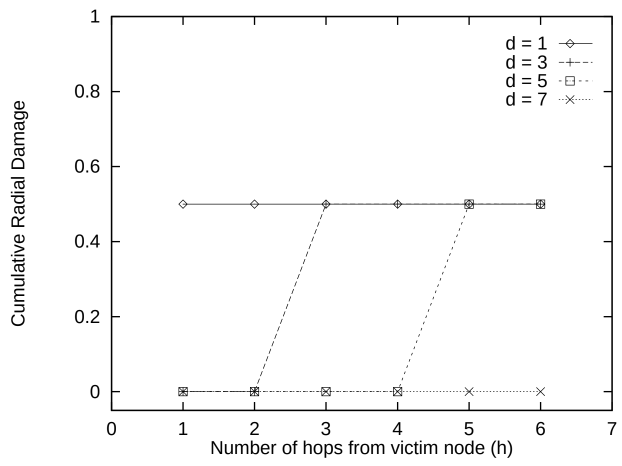

Figure 4: Damage Distribution for a Cycle with Fractional/Equal IAS/DS

Looking at Figure 4, we can see that by switching from a Weighted/Proportional IAS/DS to a Fractional/Equal IAS/DS, we are able to avoid flood damage, and we are only left with structural damage. In particular, when the Equal DS policy is used, the good node that is one hop away processes one of the victim's queries for each of the malicious node's queries before it uses all its remaining query bandwidth to service additional queries from the malicious node. Therefore, all of the victim's queries are processed at the good node, and the only damage that the victim suffers one hop away is structural.

By analyzing damage distribution, we are able to see that good policies (in particular, the Fractional/Equal IAS/DS) are able to contain flood damage. However, other mechanisms need to be developed to contain structural damage. Malicious nodes need to be detected and disconnected to deal with structural damage.

6. RELATED WORK

Most denial-of-service research to date has focused on network-layer attacks [16, 33, 34, 22, 32, 7, 29, 8, 1, 19, 9, 18, 2, 38, 20]. There have been mutliple proposals to build IP Traceback mechanisms to manage network-layer DoS attacks including [33] and [31].

Osokine [21] proposes a Q-algorithm intended for solving traffic management problems in Gnutella, but the algorithm could also be used to address DoS attacks. Rohrs [27] proposes a simplified version of Osokine's work that has been implemented in the LimeWire Gnutella client. No evaluation has been published on either proposal.

Some of the policies we propose to use to manage query floods are similar to those that have been used in link scheduling for years [40]. Algorithms such as weighted fair queuing (WFQ) have been shown to optimally allocate a fair share of link bandwidth with respect to weights. We could use WFQ to manage query flow in Gnutella nodes, but we would still need to decide on how to choose weights to minimize the damage from DoS attacks. The IASs that we use in our work can be viewed as choosing different weights for incoming query flows.

A lot of security-related research that has taken place in the P2P area has focused on providing anonymity to users [26, 6, 14] and ensuring fair resource allocation via micropayments or reputation schemes [15, 12, 25]. Other research in the area of P2P systems has focused on efficient search, routing, and indexing [35, 24, 28, 39, 5, 4, 36, 37].

7. CONCLUSION

Gnutella networks are highly susceptible to application-layer, flooding-based DoS attacks if good load balancing policies are not employed by nodes on the network. In this paper, we have taken a first step towards defining a model and metrics to measure the damage that a malicious node can cause with query flooding. Through simulations on small representative networks, we determined how damage can be minimized with load balancing policies, how damage varies as a function of network topology, and how damage is distributed.

8. REFERENCES

[1] Cert advisory ca-2000-01 denial-of-service developments. http://www.cert.org/ advisories/ CA-2000-01.html, January 2000.

[2] Edward Amoroso. A policy model for denial of service. In Proc. Computer Security Foundations Workshop III, pages 110–114, Franconia, NH USA, June 1990. IEEE Computer Society Press.

[3] Icmp traceback messages. http://www. silicondefense. com/ research/ itrex/ archive/ tracing-papers/ draft-bellovin-itrace-00.txt.

[4] Arturo Crespo and Hector Garcia-Molina. Routing indexes for peer-to-peer systems. Technical report, Stanford Univ., CS Dept., 2001.

[5] David Ratajczak, Dahlia Malkhi, Moni Naor. Viceroy: A scalable and dynamic lookup network. Proc. ACM PODC '02, August 2002.

[6] R. Dingledine, M. Freedman, and D. Molnar. The free haven project: distributed anonymous storage service.

Proc. of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA. Springer: New York (2001)., 2001.

[7] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. In IETF RFC 2267, 1998.

[8] Lee Garber. Denial-of-service attacks rip the internet. Computer, pages 12-17, April 2000.

[9] E.A. Hyden. Operating system support for quality of service. Ph.D. Thesis, University of Cambridge, 1994.

[10] Kazaa home page. http://www.kazaa.com/.

[11] Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. Secure overlay services. In *Proc. of the ACM SIGCOMM Conference, August 2002.*

[12] R. Lethin. Reputation. In Peer-to-peer: Harnessing the power of disruptive technologies. ed. Andy Oram, O'Reilly and Associates, March 2001.

[13] Limewire home page. http://www.limewire.com/.

[14] Aviel D. Rubin, Marc Waldman, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security*, August 2000.

[15] Mojo nation technical overview home page. http://www.mojonation.net/ docs/ technical_overview.shtml.

[16] D. Moore, G. Voelker, and S. Savage. Inferring internet denial of service activity. In *Proc. 2001 USENIX Security, Washington D.C., August 2001.*

[17] Morpheus home page. http://www.musiccity.com.

[18] R. M. Needham. Denial of service. In Proc. 1st ACM CCS, pg 151–153, Fairfax, Virginia, November 1993.

[19] Roger M. Needham. Denial of service: an example. *Comm. of the ACM*, 37(11):42–46, 1994.

[20] Peter G. Neumann. Inside risks: denial-of-service attacks. *Comm. of the ACM*, 43(4):136–136, 2000.

[21] Flow control algorithm for distributed 'broadcast-route' networks with reliable transport links. http://www.grouter.net/ gnutella/ flowcntl.htm.

[22] T. Ptacek and T. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., January 1998.

[23] N. Daswani and H. Garcia-Molina. Query-flood DoS Attacks in Gnutella Networks (Extended Version). Technical Report, *Stanford Univ. CS Dept.*

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. ACM SIGCOMM, 2001.

[25] Reputation technologies home page. http://reputation.com.

[26] R. Dingledine. The free haven project: Design and deployment of an anonymous secure data haven. MIT Masters Thesis May 2000.

[27] Sachrifc: Simple flow control for gnutella. http://www.limewire.com/ developer/ sachrifc.html.

[28] A. Rowstron, P. Druschel. Pastry: Scalable distributed object location and routing for largescale peer-to-peer systems. In Proc. IFIP/ACM Middleware, Heidelberg, Germany, November 2001.

[29] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society Press, May 1997.

[30] Ultrapeers: Another step towards gnutella scalability. http://groups.yahoo.com/ group/ the_gdf/ files/ Proposals/ Ultrapeer/ Ultrapeers.html.

[31] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based ip traceback. In *Proc. of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 2001.*, 2001.

[32] O. Spatscheck and L. L. Peterson. Defending against denial of service attacks in scout. In *Operating Systems Design and Implementation*, pages 59–72, 1999.

[33] A. Karlin, S. Savage, D. Wetherall, and T. Anderson. Network support for ip traceback. In *ACM/IEEE Transactions on Networking, 9(3), June 2001.*

[34] A. Karlin, S. Savage, D. Wetherall and T. Anderson. Practical network support for ip traceback. In *Proc. 2000 ACM SIGCOMM Conference, Stockholm, Sweden, August 2000.*

[35] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Technical Report TR-819, MIT, March 2001.

[36] B. Yang and H. Garcia-Molina. Designing a super-peer network. Submitted for publication.

[37] Beverly Yang and Hector Garcia-Molina. Efficient search in peer-to-peer networks. Technical report, Stanford Univ., CS Dept., 2001.

[38] C. Yu and V. Gligor. A formal specification and verification method for the prevention of denial of service. In Proc. 1988 IEEE Symposium on Security and Privacy, pages 187–202, Oakland, CA. IEEE Computer Society Press. 117, 1988.

[39] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.

[40] L. Peterson and B. Davie. Computer Networks: A Systems Approach. Morgan Kaufman: SF (2000).

APPENDIX

A. OPTIMAL RHO PROOF

THEOREM A.1. Optimal Rho ($\hat{\rho}$) for Symmetric Networks. *Suppose that for all nodes $j \in V$ have $c_j = C$ for some constant $C > 0$, $|D(j,\tau)| = D$ for some constant $D > 0$, and all nodes have ρ set to the same value, then $\hat{\rho} = 1/(D+1)$.*

PROOF. Let $\hat{\rho}$ be the setting of ρ that maximizes the total remote work. We assume that the local work processed at a node is exactly (and no less than) ρC. The maximum possible amount of remote work that can be processed at that node is $(1-\rho)C$. In the steady state, each node j is sent at most $|\delta(j,i)|\rho C$ queries that were generated i time steps ago at a node that is i hops away, $1 \leq i \leq \tau$. Hence, the maximum total amount of work that may arrive at a node is $\Sigma_{i=1}^{\tau}|\delta(j,i)|\rho C = |D(j,\tau)|\rho C = D\rho C$.

We say that a node is saturated if the amount of remote work it receives exceeds $(1-\rho)C$. A node receives less than $D\rho C$ remote work if some of the nodes that it receives work from are saturated.

No node can be saturated until the point when the maximum possible load $D\rho C$ is greater than $(1-\rho)C$, i.e. until $\rho > 1/(D+1)$. Thus, if $\rho < 1/(D+1)$, all nodes actually receive the maximum possible load, and they all become saturated at $\rho = 1/(D+1)$. For $\rho > 1/(D+1)$, the remote work is limited by $(1-\rho)C$, so $\rho = 1/(D+1)$ is the optimal value. $\square$