Preventing DoS Attacks in Peer-to-Peer Media Streaming Systems

William Conner, Klara Nahrstedt, Indranil Gupta
Department of Computer Science, University of Illinois, Urbana, IL USA 61801-2302
{wconner,klara,indy}@uiuc.edu

ABSTRACT

This paper presents a framework for preventing both selfishness and denial-of-service attacks in peer-to-peer media streaming systems. Our framework, called Oversight, achieves prevention of these undesirable activities by running a separate peer-to-peer download rate enforcement protocol along with the underlying peer-to-peer media streaming protocol. This separate Oversight protocol enforces download rate limitations on each participating peer. These limitations prevent selfish or malicious nodes from downloading an overwhelming amount of media stream data that could potentially exhaust the entire system. Since Oversight is based on a peer-to-peer architecture, it can accomplish this enforcement functionality in a scalable, efficient, and decentralized way that fits better with peer-to-peer media streaming systems compared to other solutions based on central server architectures. As peer-to-peer media streaming systems continue to grow in popularity, the threat of selfish and malicious peers participating in such large peer-to-peer networks will continue to grow as well. For example, since peer-to-peer media streaming systems allow users to send small request messages that result in the streaming of large media objects, these systems provide an opportunity for malicious users to exhaust resources in the system with little effort expended on their part. However, Oversight addresses these threats associated with selfish or malicious peers who cause such disruptions with excessive download requests. We evaluated our Oversight solution through simulations and our results show that applying Oversight to peer-to-peer media streaming systems can prevent both selfishness and denial-of-service attacks by effectively limiting the download rates of all nodes in the system.

1. INTRODUCTION

Multimedia file sharing in peer-to-peer (p2p) systems constitutes a large portion of current Internet traffic. In some cases, particularly on college campus networks, the volume of multimedia file sharing traffic has surpassed Web traffic [22]. In addition to peer-to-peer file sharing, another growing trend is that many popular Web sites have video content available for streaming [23,24]. As p2p system architectures and streaming media over the Internet both continue to gain popularity, it should be no surprise that some researchers are now designing p2p media streaming systems like PROMISE and CoopNet, which allow a single receiver to receive streams directly from multiple senders [1,2]. Unlike p2p file sharing, p2p media streaming has real-time requirements and also does not require the user to download the entire file before playback [1]. Streaming video reduces both the initial startup delay and the space requirements that downloading peers would experience in p2p file sharing systems. The p2p design distributes the workload amongst a large number of peers rather than requiring the cost of setting up a few dedicated video servers to handle the workload.

Although p2p media streaming systems offer an opportunity for peers to stream content to one another, these systems also introduce a security vulnerability. In both PROMISE and CoopNet, a receiver is free to request media streams from senders who are unaware of the total amount of bandwidth that the receiver is consuming throughout the entire system [1,2]. The multiple streams requested by a receiver could correspond to different media objects, duplicates of the same media object, or different substreams of the same media object. Ideally, a sender should deny a request to a receiver that is already consuming too much bandwidth due to selfishness or maliciousness. However, a sender only knows the amount of bandwidth that it grants to a receiver. A sender cannot determine the amount of bandwidth that the receiver is consuming from other nodes in the system. Since receivers can make unrestricted requests in such a system, opportunities exist for receivers to be malicious and selfish by requesting a large number of media streams with high data rates from different senders. If many nodes behave selfishly by consuming too much bandwidth, then well-behaving nodes might not be able to access media streams that would otherwise be available if all nodes were well-behaved. If one or more nodes exhaust all of the available upload bandwidth in the p2p media streaming system due to malice, then a denial-of-service (DoS) attack has occurred on the system because other well-behaving nodes will be prevented from downloading media streams.

One straightforward solution to this problem would be to maintain a central database that stores the current aggregate download rate of each node in the system as well as the maximum download rate allowed for each node in the system. Before granting a request, each sender could query this central database to verify that streaming the requested content to the receiver will not allow the receiver to exceed its download rate limit. This approach is illustrated in Figure 1(a). A solution similar to this centralized approach is the PlanetLab Central (PLC) infrastructure service that enforces global resource utilization for an overlay testbed [19,30]. The PLC maintains a central database of principles, slices, resource allocations, and policies. For an overlay testbed with 579 nodes (the number of PlanetLab nodes at the time this paper was written), it might be acceptable to use a single database server to enforce global resource utilization [21]. Unfortunately, a central database would not be appropriate for preventing DoS attacks and selfishness in a large p2p media streaming system with thousands of nodes because a single database server creates a bottleneck, creates a central point of failure, and lacks scalability. Therefore, a more decentralized approach will be necessary for large p2p systems.

The Oversight framework presented in this paper is a decentralized solution to the problem of preventing DoS attacks and selfishness in p2p media streaming systems. It accomplishes this prevention by enforcing maximum download rate limits for each participating node. In Oversight, a subset of trusted nodes from the p2p media streaming network organize themselves into a separate p2p network (this network will be referred to as an Oversight overlay network consisting of Oversight nodes that run the Oversight protocol). The nodes in the Oversight network will collectively store download rate information about each node in the p2p streaming network. Each Oversight node is responsible for responding to queries from potential senders about whether or not that sender should stream to a particular receiver given the receiver's current aggregate download rate and download rate limit. This will allow potential senders to use Oversight nodes to verify that they can grant a request to a receiver without allowing that receiver to violate its download limit.

Oversight nodes are also participants in the underlying p2p streaming overlay (i.e., Oversight nodes can also upload and download media objects). Unlike a solution that uses a central database server to answer node download queries, the Oversight framework provides an approach that avoids a central bottleneck, avoids a central point of failure, and provides scalability. Figure 1 below contrasts the two approaches. The numbered boxes are p2p media streaming nodes and the arrows represent queries and responses. Note that shaded nodes 2, 5, and 7 in Figure 1(b) are all Oversight nodes. This means that these nodes run the protocols associated with both the underlying media streaming overlay network *and* the Oversight overlay network.

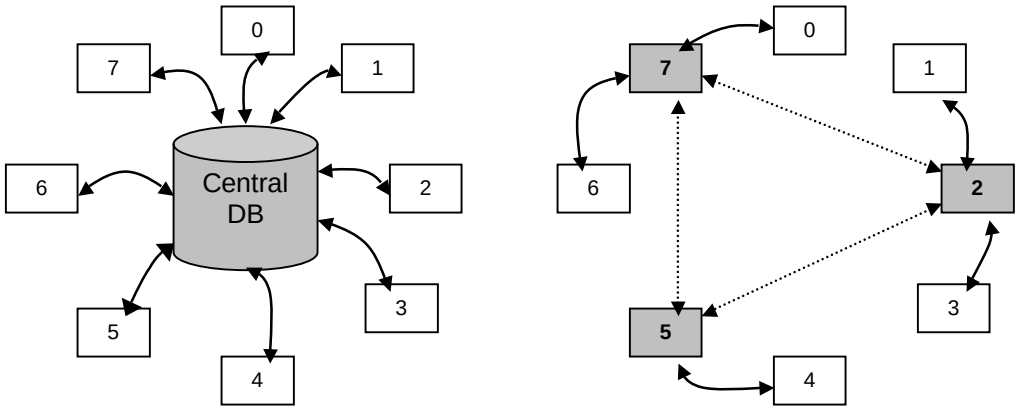


Figure 1. Comparison of central database server architecture versus Oversight framework

The next section of this paper discusses related work. After that, we introduce our complete Oversight framework in Section 3. In section 4, we evaluate the performance of the Oversight framework by presenting our simulation results. Finally, we conclude in the last section.

2. RELATED WORK

Before discussing previous work in distributed systems security at the network and application layers, it is important to understand the p2p media streaming systems that motivated the monitoring approach presented in this paper. Hefeeda et al. present a p2p media streaming system called PROMISE, which is built on top of their p2p service named CollectCast [1]. PROMISE supports peer-based aggregated streaming where multiple senders collectively stream a media object to a single receiver. In PROMISE, the receiver queries with the underlying p2p lookup protocol, such as Chord [13] or Pastry [14], to locate peers that store a copy of the desired object. Maintaining a candidate set of sending peers based on query results, the receiver will select a subset of peers from the candidate set to send parts of the media object at various assigned rates. Padmanabhan et al. present a p2p media streaming system called CoopNet that is only used when the primary video server experiences overload [2]. Such overload conditions often occur during flash crowds (see [6] for a good discussion of flash crowd characteristics). In CoopNet, the video server redirects requests for media objects to clients that have recently downloaded that object. Using multiple description coding, each substream from one of many senders is sent to a single receiver. Application-layer multicast solutions like the one presented in [28] also use p2p media streaming, but these solutions usually go from one sender to many receivers rather than many senders to one receiver. In all of these p2p media streaming systems, a selfish or malicious peer could potentially exhaust the resources available in the system by aggressively requesting a large number of high quality streams in a short amount of time.

Much previous work has been done on security at the network layer, particularly in the prevention of network-layer DoS attacks. Ioannidis and Bellovin present a "pushback" approach where routers use heuristics to try to distinguish between packets belonging to legitimate flows and packets that are part of an attack [7]. When bad flows are identified at a particular router, it notifies upstream routers to rate-limit the bad traffic, hence the name "pushback." Park and Lee propose a different approach at the network layer where routers filter packets based on their source and destination addresses [9]. Basically, if an address pair seems invalid for a particular router (e.g., a packet going from source *A* to destination *B* should not go through router *C*), then the packet is discarded. Such an approach defends against distributed DoS attacks because spoofed IP packets are filtered. Sekar et al. combine the strengths of both anomaly-based and specification-based techniques to develop a better network intrusion detection system [17]. Talwar and Nahrstedt use two different types of message integrity checks (one for messages within the same subnetwork and another for messages that go across subnetworks) to help prevent some of the potential DoS attacks on RSVP, a protocol for setting up end-to-end resource reservations [29].

Research has also been done on preventing DoS attacks at the application layer. Dean and Stubblefield propose a new approach to client puzzles to protect TLS, a protocol for providing confidentiality and integrity between two communicating endpoints [10]. In their approach, cryptographic puzzles must be solved by all clients making requests in order to slow down potential DoS attackers. Kruegel and Vigna propose anomaly detection techniques for Web servers that compare HTTP query request logs with established profiles [16]. Jin et al. describe a technique for Web servers to detect and discard requests from spoofed IP addresses through the use of hop count filtering [8].

Much work has also been done on security at the application layer with p2p systems. Daswani et al. have identified DoS attacks as one of the major open problems in p2p systems [4]. Douceur points out that any redundancy mechanisms in p2p systems that are intended to increase protection from malicious attackers will not be effective unless there is a way to ensure that a single peer cannot present itself as being multiple peers [3]. The attack described in [3] is commonly referred to as a Sybil attack. In general, replication is not enough to prevent such attacks. For p2p systems that use distributed hash tables (DHTs), Sit and Morris survey many of the various attacks that could take place [18]. The attacks range from routing attacks to storage and retrieval attacks. Castro et al. have introduced some techniques to defend against routing attacks in p2p systems when the number of malicious nodes is limited [15]. Daswani and Garcia-Molina have suggested that query flooding attacks in the unstructured Gnutella p2p system can be prevented by having each node fairly allocate its processing capacity to requesting peers during overload [11]. Keyani et al. propose that nodes in a Gnutella network keep backup topology information so that a topology more resilient to DoS attacks (i.e., exponential network topology) can be used in the event that an attack occurs on the regular power law topology [12].

As mentioned already in Section 1, the PLC infrastructure service uses a centralized approach for enforcing global resource utilization in PlanetLab [30]. An alternate proposal for distributed resource management intended for use with PlanetLab is SHARP [31]. In SHARP, peers acquire resources by exchanging resource claims with one another in a decentralized fashion. However, one problem with applying the SHARP framework to a large p2p network is that a peer must first acquire a resource claim before accessing a resource at another peer. So, in addition to locating a resource, a peer would also have to separately locate a resource claim to use that resource in the SHARP framework.

This resource claim acquisition step might be prohibitively expensive in large p2p networks depending on the length of the chain of peer relationships between the requester and resource claim holder.

3. OVERSIGHT FRAMEWORK

Given the prevalence of DoS attacks in the Internet, as shown by the study in [5], it is important to realize that a similar threat could exist in a large-scale p2p media streaming system where attackers are capable of consuming a lot of resources with just a small amount of effort (i.e., sending a few requests that take a small amount of bandwidth could lead to receiving several high bandwidth streams that exhaust resources). In this section, we will describe the application model, p2p lookup model, and attack model that we assume for our solution. Then, we will present our solution to the prevention of DoS attacks in p2p media streaming systems using the Oversight framework.

3.1. Application Model

Informally, our p2p media streaming application consists of a set of distributed nodes that stream media objects to one another. Media objects are identified and located using the p2p lookup model described in Section 3.2. For the purposes of our general model, the media object type and unit are left unspecified. For specific applications, a media object could be one of any media types (e.g., audio, video) in one of any unit types (e.g., partial segment, whole clip). Each node stores zero or more media objects. Nodes upload and download media objects to and from other nodes. Our model assumes that every node has a maximum upload rate and a maximum download rate. These maximum rates are also application-specific as they might be determined by either policy (e.g., an application might prohibit any node from downloading over 1 Mbps) or physical limitations (e.g., a node using a dial-up connection might only be able to download at 56 Kbps).

A formal description of the model appears in Tables 1 and 2 below.

P = set of nodes in the system
n = number of nodes in the system = $\|P\|$
m_i = number of media objects stored at node i
a_{ijk} = 1 if node i is uploading its k^{th} media object to node j, 0 otherwise
b_{ik} = data rate of k^{th} media object stored at node i
r_{ij} = current data rate from node i to node j = $\sum_{k=1 \text{ to } m(i)} (a_{ijk} \cdot b_{ik})$
$Dmax_j$ = maximum download rate allowed for node j
D_j = current aggregate download rate for node j = $\sum_{i=1 \text{ to } n} (r_{ij})$
$Umax_i$ = maximum upload rate allowed for node i
U_i = current aggregate upload rate for node i = $\sum_{j=1 \text{ to } n} (r_{ij})$

Table 1. Formal application model definitions

$D_j \leq Dmax_j$ for each node j
$U_i \leq Umax_i$ for each node i

Table 2. Formal application model constraints

Referring to the model above, for some node *i*, enforcing the constraint that U_i not exceed $Umax_i$ or the constraint that D_i not exceed $Dmax_i$ are relatively simple because each node can locally determine how much it is uploading or downloading. However, nodes only have motivation to enforce the constraint concerning their upload limits because they do not benefit from servicing others' requests in our model. Nodes have no motivation to enforce the constraint concerning their own download limit because they would be limiting their own benefit. Therefore, download rate limitations must be enforced by some other means that does not depend on nodes limiting themselves individually. Oversight provides a solution to this problem.

3.2. P2P Lookup Model

Our application model assumes an underlying p2p routing substrate that uses a ring-like distributed hash table (DHT) with an *m*-bit circular identifier space, such as Chord [13] or Pastry [14]. The reason that a ring-like DHT-based p2p lookup protocol was chosen is due to the fact that such protocols support distributed lookup in an efficient and scalable manner.[1] In these lookup protocols, the messages in a system with *N* nodes take O(log *N*) application-layer hops to reach their destination and each node stores O(log *N*) entries in its routing table. This allows peers to perform decentralized searches and locate objects with relatively few message hops and a small amount of storage needed for routing information. Both Chord and Pastry are also self-organizing and can adapt automatically even when nodes join and leave. PROMISE, one of the p2p media streaming systems discussed in Section 2, also happens to be built on top of Pastry.

Before discussing our solution, it is necessary to provide some background information on how lookups are performed in a ring-like DHT-based system. The basic service provided to nodes by p2p lookup protocols is object location. Both nodes and objects are assigned unique identifiers (usually called keys for objects and identifiers for nodes) from the same circular identifier space. Each object's lookup key is mapped to a unique node. The assignment of object keys to nodes is based on the object key and node identifier. Different p2p lookup protocols have different algorithms for object key assignment. For example, Pastry maps object keys to the numerically closest node identifier while Chord maps each object key to the numerically closest node identifier that is equal to or succeeds the object key in the circular identifier space. Figure 2 illustrates how the following object keys would be assigned in either protocol assuming 4-bit identifiers: 0x3, 0x4, 0x5, 0x9, 0xF.

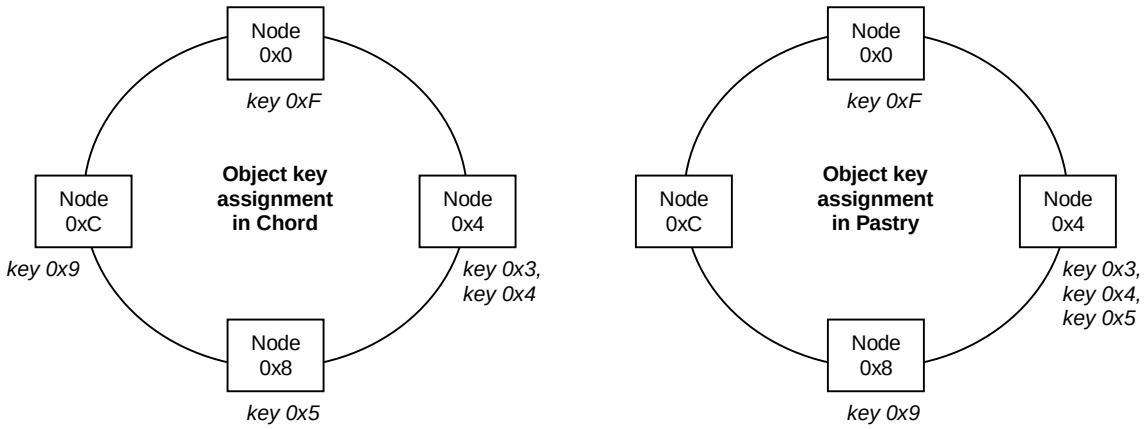


Figure 2. Object key assignment

In order to locate an object, a node will initiate a distributed lookup according to the specific p2p routing substrate algorithm. A query message (or object key lookup) takes O(log *N*) application-layer hops from source to destination in [13,14]. Also, in a system with *N* nodes, each node has a routing table with O(log *N*) entries where each

[1] The discussion in Section 3.2 is specific to ring-like DHT-based p2p lookup protocols and was included for clarity in later sections when discussing the Oversight framework. However, the ideas presented later in this section can also be used with ring-less DHT-based p2p lookup protocols like Kelips [27], which has different object key lookup performance and storage requirements at each node compared to the ring-like p2p routing substrates presented in [13,14].

entry maps a node identifier to an IP address and port number. Using the routing table, each intermediate node along the routing path will forward the message to the *best* node in its routing table among all the candidate nodes stored as routing table entries. Here, the *best* node in the routing table is specific to the particular routing algorithm.[2]

3.3. Attack Model

Using our application model as a reference, a selfish or malicious node *j* could make several requests to different nodes such that D_j is greater than $Dmax_j$, but for each node *i* uploading to node *j*, $r_{ij} \leq Dmax_j$. Such an attack is shown in Figure 3. Although each uploading node *i* can determine the rate that it has granted to a particular node *j* (i.e., determine r_{ij}), it cannot determine node *j*'s overall download rate (i.e., cannot determine D_j). Therefore, such an attack could succeed unless there is some way for potential uploading nodes to query about potential downloading nodes' current aggregate download rates in relation to their maximum download rate allowed (i.e., each node *i* must determine D_j and $Dmax_j$ before uploading to node *j*).

There are several reasons that a node might decide to attempt to exceed its download rate. For example, a selfish node might exceed its download rate to achieve better quality (e.g., downloading additional layers in a multiple description coding scheme like that presented in [2]). Or, a malicious node might exceed its download rate to be disruptive and cause a DoS attack. The Oversight protocol is designed in such a way that when a node *i* receives a request from node *j* for a media object, it can perform a query to determine whether granting the request will cause D_j to exceed $Dmax_j$.

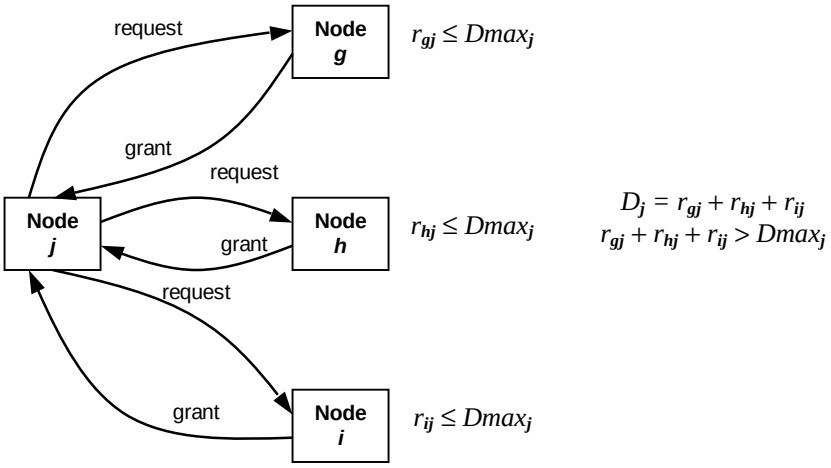


Figure 3. Example attack executed by node *j*

Since our attack model only considers attacks executed by downloading nodes, the Oversight framework focuses primarily on preventing attacks from selfish or malicious downloading nodes. Some potential attacks executed by uploading nodes on the Oversight protocol are briefly discussed in Section 3.4.3. The expansion of the Oversight framework to prevent those attacks from uploading nodes will be part of our future work.

3.4. Oversight Protocol

The main idea behind the Oversight protocol is that selfishness and DoS attacks in p2p media streaming systems can be prevented by having a subset of trusted nodes form a separate overlay that stores node download information objects (i.e., stores D_j and $Dmax_j$ for each node *j* in the system). These trusted nodes will be referred to as Oversight nodes and the remaining nodes will be referred to as regular nodes. Our protocol assumes that each regular node knows the IP address of the Oversight node that stores its download information (i.e., each regular node knows how to contact one

[2] For the purpose of discussing the Oversight framework, the specific routing algorithm details are not important. Please refer to [13] or [14] to get more details about ring-like DHT-based p2p routing algorithms.

Oversight node). Treating node identifiers from the streaming application overlay as node download information object keys in the Oversight overlay, D_j and $Dmax_j$ for each node j will be stored at the Oversight node to which key j maps according to the p2p lookup protocol. Before a node i can stream a media object to some node j, it must query its known Oversight node o about whether or not the request should be granted (assuming that node i is not itself an Oversight node). Oversight node o will retrieve D_j and $Dmax_j$ from either its local database of node download information objects or retrieve those values by requesting the node download information object from the Oversight node o' responsible for storing key j. The underlying p2p lookup protocol can be used to locate and retrieve these objects. Upon receiving D_j and $Dmax_j$, Oversight node o will only give permission to node i to stream the media object to node j if the additional stream's data rate will not cause D_j to exceed $Dmax_j$. Some object-oriented pseudocode for the Oversight protocol appears in the Appendix. Figure 4 below describes how the Oversight protocol would work if some node j requested a media object from some regular node i.[3] If node j requested a media object stored at an Oversight node, then the steps in the middle of Figure 4 would be omitted.

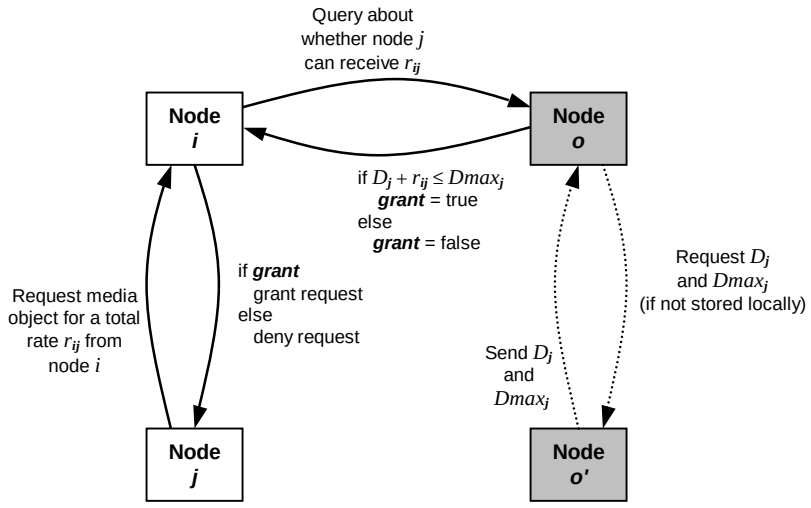


Figure 4. Oversight protocol

The following concrete example illustrates the operation of the Oversight protocol with a small network consisting of eight regular nodes that upload and download media objects to and from one another. In this example, keys are assigned to nodes according to Pastry's object key assignment for a 3-bit circular identifier space as discussed in Section 3.2. In the following example, we will go through the steps involved for a node with identifier 0x4 to receive a stream of a media object whose key is 0x3. Three of the regular nodes below are also Oversight nodes as indicated by the shaded nodes in Figure 5.

Example protocol steps:

1. Node 0x4 sends request for media object 0x3.
2. Node 0x3 sends query to its known Oversight node 0x2 about whether or not it should upload media object 0x3 to node 0x4.
3. Node 0x2 sends query for node 0x4's download info to node 0x5 that stores download info for node download information object key 0x4.
4. Node 0x5 sends node 0x4's download info to node 0x2.
5. Node 0x2 tells node 0x3 whether or not it should stream media object 0x3 to node 0x4.
6. Node 0x3 either denies or grants node 0x4's request for media object 0x3 based on the decision from node 0x2.

[3] Some steps are omitted for clarity. Please refer to the Appendix for more details.

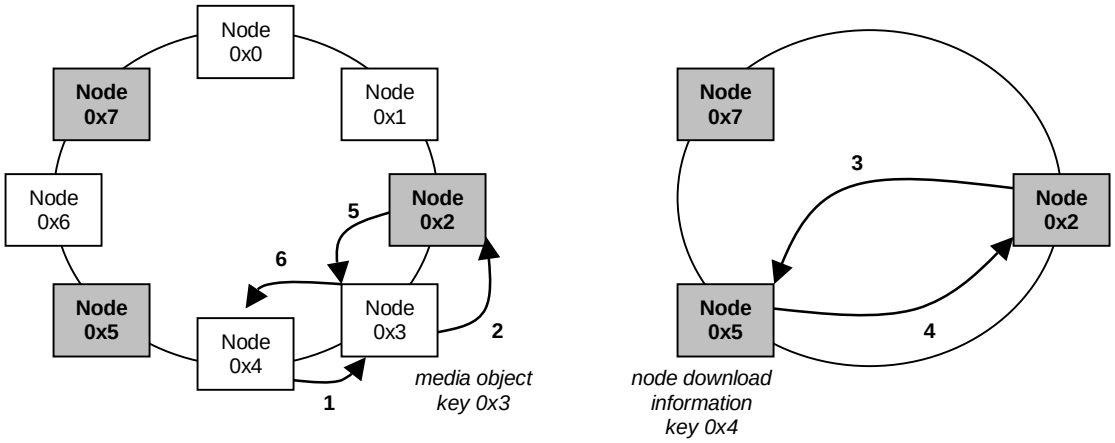


Figure 5. Concrete Oversight protocol example

3.4.1. Trusted Peer Nodes

An important assumption made in the Oversight protocol is that a separate overlay of trusted nodes can be established to act as Oversight nodes. One question about such an assumption is how can one determine whether or not a node can be trusted to be an Oversight node. Although this paper focuses on building a protocol under the assumption that a subset of nodes can be trusted, we will briefly mention some ideas on how trust can be established.

The first idea is that the p2p applications can provide two methods of authentication with one method for trusted nodes and another method for untrusted nodes. Authenticating trusted nodes, of course, would involve additional requirements compared to the requirements for authenticating an untrusted node. For example, maybe the owner of a trusted node has to fill out a paper application identifying possible users with their contact information while the owner of an untrusted node only has to join the system anonymously. In this case, a trusted node has less incentive to misbehave since users are more likely to be held accountable if caught. Additional requirements for trusted nodes could be enforced (e.g., background checks indicating no computer abuse in the past or requiring that the trusted node be owned by a university). Incentives, such as higher download rate maximums, could be offered to nodes that are willing to participate as trusted nodes. This would ensure that enough nodes serve as Oversight nodes.

Another idea is that a node could be trusted only if it builds a reputation that surpasses a certain threshold by adopting techniques from one of the reputation-based p2p systems [25,26]. Using a reputation-based approach, a single node (possibly owned by the company distributing the p2p streaming application) could be the first Oversight node. As time goes on, the Oversight node could collect information that would allow it to determine whether or not other untrusted nodes could be upgraded to trusted node status in order to participate in the Oversight overlay. Of course, a possible vulnerability that must be defended against is for an attacker to initially build up a reputation through good behavior for some time and then misbehave once it reaches Oversight node status. Establishing trusted nodes in p2p systems based solely on reputation is a difficult problem and will be part of our future work.

3.4.2. Message Authentication and Delivery

Two additional security problems that occur within Oversight are that messages could be dropped or modified in unauthorized ways by malicious nodes. If attackers drop messages, then they might prevent a well-behaving node from downloading some media object because the Oversight protocol cannot complete. In this situation, a node processing a download request will not make any progress on that request because it will not receive a response to its query about the well-behaving node's download rate and download limit. If attackers alter messages, then they could underreport their download rates in order to consume more bandwidth. Both problems can be addressed by using some of the techniques

presented in [15]. These techniques include using a certificate authority (CA) to distribute random ID assignments and public keys. Both the random ID assignments and public keys have certificates signed by the CA. Random ID assignment can be used to ensure that malicious nodes cannot collude by clustering themselves in the identifier circle of the DHT in such a way that they can partition well-behaving nodes by maliciously dropping their messages. Public key certificates can be used to authenticate messages through digital signatures. This prevents attackers from forging and sending their own Oversight reply messages in an attempt to impersonate an Oversight node. Constrained routing tables and redundant routing are also suggested in [15] to help defend against routing attacks on the distributed hash table.

Although a single CA would violate the decentralized design principles of p2p media streaming systems, it should be noted that its involvement would be minimal. Once the appropriate certificates are issued for node IDs and public keys, the participating peers can then verify certificates and digital signatures without involving the CA as long as all peers know the public key of the CA. The benefit of defending against routing attacks and providing a way to authenticate messages is crucial for secure p2p media streaming.

3.4.3. Malicious Uploading Nodes

One possible attack on the Oversight protocol is that a malicious node receiving a download request could run the protocol indicating that it has agreed to upload the requested media file to the requester and then not actually upload the file. This would increase the current download rate associated with the requester's node download information object, even though the requester is not actually receiving the requested media object. When the requester tries to seek the media object elsewhere, their request might be denied because the additional request might cause them to exceed their maximum download rate, even though they are not actually exceeding their download limit since the malicious node is not actually uploading the media file.

One possible solution to avoid this problem would be to use a reputation scheme for p2p systems, such as the protocol presented in [25]. Damiani et al. developed a polling protocol for Gnutella-like p2p systems that allows downloading nodes to choose uploading nodes and resources based on their reputations according to other nodes in the system [25]. The approach in [25] could be modified for ring-like DHT-based p2p protocols so that requesters could avoid requesting media files from malicious uploading nodes that have developed poor reputations over time. Incorporating reputations into the Oversight framework is part of our future work.

4. PERFORMANCE EVALUATION

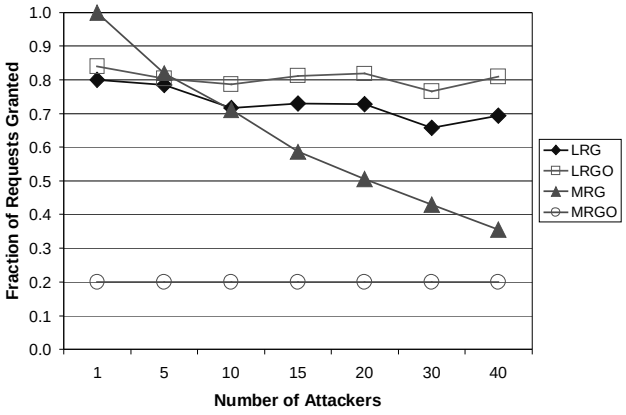
In this section, we evaluate the performance of the Oversight protocol with respect to its effectiveness at preventing DoS attacks. To evaluate the performance of our protocol, we ran two types of simulations of a p2p media streaming application (as described in Section 3.1): one type where the Oversight protocol was used and another type where the Oversight protocol was not used. These two types of simulations gave us a basis for comparison. Our simulations were implemented on top of FreePastry [20]. In each simulation, we observe the fraction of requests granted to benign and malicious nodes when the Oversight protocol is used and compare that to the fraction of requests granted to benign and malicious nodes when the Oversight protocol is not used. When the Oversight protocol is not used, each node decides whether or not to grant a request based on whether its upload limit will be exceeded. When the Oversight protocol is used, each node runs the Oversight protocol as described in Section 3.4. In our simulations, we expected to observe that using the Oversight protocol causes a larger fraction of legitimate requests from benign nodes to be granted compared to when the Oversight protocol is not used. Conversely, we expected to observe that using the Oversight protocol causes a larger fraction of malicious requests from attackers to be denied compared to when the Oversight protocol is not used.

In our simulations, we assume that each media object requested has a playback duration of three minutes and has a data rate of 350 Kbps. We chose to give each node an upload limit of 1 Mbps and a download limit of 1 Mbps. Each benign node makes between one and five total requests for media objects during each simulation. These requests are spaced between three and six minutes apart. The number of requests from each benign node, the spacing between requests, and the media object keys requested are all random values (e.g., the total number of requests from a benign node will be a random integer between one and five). Each attacker (i.e., malicious node) makes 10 requests for random media object keys that are spaced 100 milliseconds apart. Based on these specified behaviors in our simulation, no benign node will attempt to exceed its download limit and every attacker will attempt to exceed its download limit.

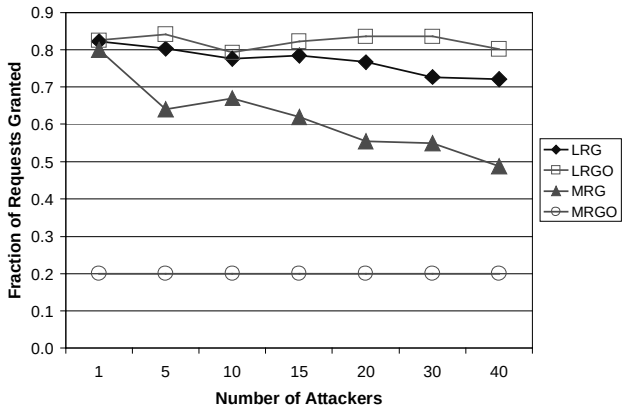
For the results that appear in Figure 6(a), the p2p network had a total of 100 nodes with 10 of those 100 nodes being Oversight nodes. For the results that appear in Figure 6(b), the p2p network had a total of 200 nodes with 10 of

those 200 nodes being Oversight nodes. In both cases, we varied the number of attackers in our simulations to see how they would affect the fraction of requests granted for the following categories:

LRG = fraction of legitimate requests granted to benign nodes without Oversight
LRGO = fraction of legitimate requests granted to benign nodes with Oversight
MRG = fraction of malicious requests granted to attackers without Oversight
MRGO = fraction of malicious requests granted to attackers with Oversight



(a) 100 total nodes, 10 Oversight nodes



(b) 200 total nodes, 10 Oversight nodes

Figure 6. Simulation results

As Figure 6 shows, a larger fraction of requests are granted to benign nodes when Oversight is used (i.e., LRGO) compared to when Oversight is not used (i.e., LRG). These are the results that we would expect given the behavior of benign nodes and attackers in the simulation. More importantly, Figure 6 shows that the Oversight protocol severely limits malicious requests from attackers (i.e., MRGO) compared to when Oversight is not used (i.e., MRG). The reason that each attacker is granted 20% of its requests in both simulations is because each attacker is able to download the first two media objects that it requests, but cannot download its remaining eight requests. The reason that the first two requests (spaced 100 milliseconds apart) from attackers are granted is because a node can download two media objects simultaneously at a combined rate totaling 700 Kbps (each media object is 350 Kbps) and stay within its download limit of 1 Mbps. However, the remaining requests (also space 100 milliseconds apart) would cause an attacker's download rate limit to be exceeded.

Our simulation results also indicate that even benign nodes cannot be granted all of their requests. This observation is due to the fact that each node has an upload rate limit of 1 Mbps that cannot be exceeded. Thus, if three benign nodes simultaneously request media objects (with each media object having a data rate of 350 Kbps) that happen to be stored at the same node, then at least one benign node will be denied its request due to upload rate limits at the node storing the media objects. The same reasoning applies for why attackers cannot be granted all of their requests even though the Oversight protocol is not used. For example, in Figure 6, the reason that the fraction of requests granted to attackers declines as the number of attackers increases is due to the fact that the bursts of requests associated with their DoS attack are more likely to compete with one another for limited upload bandwidth when there are more attackers making such requests.

Although not shown in Figure 6, we have verified that all requests denied to benign nodes in the simulations, when the Oversight protocol was used, were due to nodes storing the requested media objects reaching their upload rate limits, rather than the benign requesters being denied because some Oversight node mistakenly thought they attempted to exceed their download rate limits.

5. CONCLUSION

The main contribution of this work is that we have demonstrated that a subset of trusted nodes in a p2p media streaming overlay network can organize into a separate p2p overlay network to enforce download rate limitations in a decentralized

manner. This is the solution provided by the Oversight framework. These download rate limits prevent greedy behavior and DoS attacks from selfish and malicious nodes, respectively. The benefits of using the Oversight framework, compared to a centralized database server architecture that enforces rate limitations, is that Oversight provides a scalable solution while also avoiding the drawbacks of having a bottleneck and central point of failure.

As our simulations show, these benefits come at the cost of increased overhead at Oversight nodes that must store node download information objects and answer queries concerning these objects. However, the benefit of paying these costs is preventing selfish and malicious nodes from exhausting the resources of an entire p2p media streaming system. As our simulations show, the Oversight protocol is very effective at reducing the number of requests granted to attackers and thus preventing DoS attacks.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful comments on how to improve the final draft of this paper. This work was supported by the AT&T Labs Fellowship Program and the National Science Foundation under grant NSF ANI 03-23434. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

REFERENCES

1. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. "PROMISE: peer-to-peer media streaming using CollectCast." *11th ACM Conference on Multimedia*, 2003.
2. V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. "Distributing streaming media content using cooperative networking." *12th International Workshop on Network and Operating System Support for Digital Audio and Video*, 2002.
3. J. Douceur. "The Sybil attack." *1st International Workshop on Peer-to-Peer Systems*, 2002.
4. N. Daswani, H. Garcia-Molina, and B. Yang. "Open problems in data-sharing peer-to-peer systems." *9th International Conference on Database Theory*, 2003.
5. D. Moore, G. Voelker, and S. Savage. "Inferring Internet denial-of-service activity." *10th USENIX Security Symposium*, 2001.
6. J. Jung, B. Krishnamurthy, and M. Rabinovich. "Flash crowds and denial-of-service attacks: characterization and implications for CDNs web sites." *11th International World Wide Web Conference*, 2002.
7. J. Ioannidis and S. Bellovin. "Implementing pushback: router-based defense against DDoS attacks." *Network and Distributed Systems Security Symposium*, 2002.
8. C. Jin, H. Wang, and K. Shin. "Hop-count filtering: an effective defense against spoofed DDoS traffic." *10th ACM Conference on Computer and Communications Security*, 2003.
9. K. Park and H. Lee. "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets." *ACM SIGCOMM*, 2001.
10. D. Dean and A. Stubblefield. "Using client puzzles to protect TLS." *10th USENIX Security Symposium*, 2001.
11. N. Daswani and H. Garcia-Molina. "Query-flood DoS attacks in Gnutella." *9th ACM Conference on Computer and Communications Security*, 2002.
12. P. Keyani, B. Larson, and M. Senthil. "Peer pressure: distributed recovery from attacks in peer-to-peer systems." *NETWORKING Workshops*, 2002.
13. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. "Chord: a scalable peer-to-peer lookup service for internet applications." *ACM SIGCOMM*, 2001.
14. A. Rowstron and P. Druschel. "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems." *Middleware*, 2001.
15. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. "Secure routing for structured peer-to-peer overlay networks." *5th Symposium on Operating System Design and Implementation*, 2002.
16. C. Kruegel and G. Vigna. "Anomaly detection of Web-based attacks." *10th ACM Conference on Computer and Communications Security*, 2003.
17. R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. "Specification-based anomaly detection: a new approach for detecting network intrusions." 9th *ACM Conference on Computer and Communications Security*, 2002.

18. E. Sit and R. Morris. "Security considerations for peer-to-peer distributed hash tables." *1st International Workshop on Peer-to-Peer Systems*, 2002.
19. L. Peterson, T. Anderson, D. Culler, and T. Roscoe. "A blueprint for introducing disruptive technology into the Internet." *1st ACM Workshop on Hot Topics in Networking*, 2002.
20. FreePastry. *http://freepastry.rice.edu/FreePastry/*.
21. PlanetLab – Home. *http://www.planet-lab.org*.
22. K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. "Measurement, modeling, and analysis of a peer-to-peer file sharing workload." *19th ACM Symposium on Operating Systems*, 2003.
23. MSNBC. *http://www.msnbc.msn.com*.
24. CNN.com. *http://www.cnn.com*.
25. E. Damiani, S. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. "A reputation-based approach for choosing reliable resources in peer-to-peer networks." *9th ACM Conference on Computer and Communications Security*, 2002.
26. S. Marti and H. Garcia-Molina. "Identity crisis: anonymity vs. reputation in p2p systems." *3rd IEEE International Conference on Peer-to-Peer Computing*, 2003.
27. I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. "Kelips: building an efficient and stable p2p DHT through increased memory and background overhead." *2nd International Workshop on Peer-to-Peer Systems*, 2003.
28. Y. Cui and K. Nahrstedt. "Layered peer-to-peer streaming." *13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2003.
29. V. Talwar and K. Nahrstedt. "Securing RSVP for multimedia applications." *Multimedia Security Workshop, ACM Multimedia*, 2000.
30. B. Chun and T. Spalink. "Slice creation and management." PlanetLab Design Note 03-013, July 2003.
31. Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. "SHARP: an architecture for secure resource peering." *19th ACM Symposium on Operating Systems Principles*, 2003.

APPENDIX

class MediaObject: // Media object class definition

```
// Instance variables for MediaObject objects

MediaObjectKey key;     // Object key for this media object
int rate;               // Data rate of this media object
```

class RegularNode: // Regular p2p media streaming application node class definition

```
// Instance variables for RegularNode objects

RegularNodeId id;           // Node identifier for this node
OversightNode onode;        // Known Oversight node as described above
MediaObjectDB db;           // Local database of media objects stored at this node
int current_upload_rate;    // Amount of upload bandwidth currently being consumed
int max_upload_rate;        // Maximum upload rate allowed

// Methods that can be invoked on RegularNode objects

void recvReq(MediaObjectKey k, RegularNodeId rnid){
   // Invoked upon receiving a request for a media object with
   // key k from regular node with identifier rnid
   MediaObject mobject = db.get(k);
   if mobject is null
      // Media object with key k was not found
      then deny request for k and return;
   if current_upload_rate + mobject.rate > max_upload_rate
      // Streaming media object with key k will cause this node to
      // exceed its upload limit
      then deny request for k and return;
   else
      // Stream media object with key k only if known
      // Oversight node says that regular node with identifier rnid
      // would not exceed its download limit if it receives mobject
      boolean grant = onode.query(rnid, mobject.rate);
      if grant
         // Regular node with identifier rnid would not
         // exceed limit
         current_upload_rate = current_upload_rate +
                                          mobject.rate;
         send stream of mobject to rnid;
         end(rnid, mobject.rate);
      else
         // Regular node with identifier rnid would exceed limit
         deny request for k and return;
}

void sendReq(MediaObjectKey k){
   // Send a request for media object with key k
   RegularNode rnode = do p2p lookup
         on k to find regular node that stores k;
   Ask rnode to stream media object with
         key k to this node;
}

void end(RegularNodeId rnid, int rate){
   // Inform Oversight node that streaming
   // session has ended
   onode.end(rnid, rate);
   return;
}
```

class NodeDownloadInfo: // Node download information class definition

```
// Instance variables for NodeDownloadInfo objects

RegularNodeId rnid;   // Node identifier for corresponding node download information
int D;                // Current download rate for node rnid
int DMax;             // Maximum download rate allowed for node rnid
```

class OversightNode: // Oversight node class definition

// Note that each OversightNode instance also participates as a RegularNode, which
// refers to itself as its known *onode* from the RegularNode instance variable

// Instance variables for OversightNode objects

NodeDownloadInfoDB *db*; // Local database of node download info stored at this node

// Methods that can be invoked on OversightNode objects

boolean query(RegularNodeId *rnid*, int *rate*){
// Check whether or not regular node with identifier *rnid* can
// increase its download rate without exceeding its limit
NodeDownloadInfo *ni* = do p2p lookup on *rnid* and retrieve
node download info where *ni.rnid* equals *rnid*;
int *requesterD* = *ni.D*;
int *requesterDMax* = *ni.DMax*;
if (*requesterD* + *rate* < *requesterDMax*)
// Additional increase in download rate will not cause this
// node to exceed its download limit, so allow the increase
// and update download rate for requester
ni.D = *ni.D* + *rate*;
return true;
else
// Additional increase in download rate will cause this node
// to exceed its download limit, so do not allow the increase
return false;
}
}

void end(RegularNodeId *rnid*, int *rate*){
// Update node download information for
// node with identifier *rnid* to reflect that
// it has reduced its download rate by the
// amount *rate*
NodeDownloadInfo *ni* = do p2p lookup
on *rnid* and retrieve node download info
where *ni.rnid* equals *rnid*;
ni.D = *ni.D* – *rate*;
return;
}