The Effects of Perturbation on Gnutella Network

Steven Ko, Indranil Gupta

Department of Computer Science
University of Illinois, Urbana-Champaign
{sko, indy}@cs.uiuc.edu

Abstract

It is well known that a user, who participates in an overlay network like Gnutella, does not stay in the network continuously, but goes online and offline repeatedly. In addition to this user behavior, dynamic network conditions like congestion, routing flaps in BGP, and intermittent router failures can force a node unavailable for a short period of time. This paper studies how Gnutella would perform under those stressful network environments caused either by the network itself or by the users of the overlay network. According to our results, Gnutella tolerates node perturbations well and shows even better performance when a small portion of nodes is perturbed. This behavior of Gnutella confirms the common belief that unstructured peer-to-peer systems can tolerate dynamic changes of the overlay topology.

1 Introduction

An overlay network is an application-layer network constructed on top of existing network like the Internet. Each node participating in the overlay network runs an application which implements a routing algorithm for the overlay and is responsible for forwarding messages sent by other nodes in the overlay according to the routing algorithm. “One hop” transfer between nodes in the overlay is through the existing network and can be any number of hops in the underlying network’s point of view.

There are two types of setting in overlay networks. First, an overlay can be constructed from a static setting. Each node is statically assigned to form an overlay network. RON [7] is an example of this type of setting. They are used in scenarios like an alternative internet routing [7]. Second, each node can dynamically join or leave the overlay. Many peer-to-peer systems like Gnutella [6] are of this kind. Another classification of overlay networks is based on the types of routing algorithms. Unstructured peer-to-peer systems represented by Gnutella [6] are in the first category. In this type of systems, each node floods the overlay network of peers to find the destination of a request. Although it does not scale well due to its flooding-based nature, it is believed to have an advantage of tolerating dynamic changes of the overlay network topology as nodes join in and depart from the system. The second category is structured peer-to-peer systems. Chord [12], Pastry [3], and Kelips [13] fall into this category. This type of systems, known as Distributed Hash Tables (DHTs), transforms a peer-to-peer system into a DHT and tries to solve the problem of insertion, deletion, and lookup of the DHT. These DHTs have advantages over unstructured peer-to-peer systems, e.g. security, scalability, load-balancing, and predictable cost of insertion, deletion, and lookup. However, it has not been studied how DHTs would work in real, dynamic environment [11].

Our study stems from the fact that how these algorithms would work in dynamic environment is not known to researchers yet. So we start exploring that direction first by studying how Gnutella would perform under the dynamic environment using GnutellaSim [5], a Gnutella simulator built on top of ns-2. Basically, we measure successful replies to queries injected to the overlay network
while every node, except the nodes that send queries to the overlay, goes offline and comes back online periodically. We will refer to this periodic behavior of nodes as *flap* and the nodes which shows the behavior as *flapping* nodes. Our results show that in general, Gnutella tolerates flapping nodes well. An interesting result from our simulations is that when a small portion of nodes flaps in Gnutella network, it performs even better than the time when there’s no flapping node.

The rest of this paper is organized as follows. Section 2 presents the related work, section 3 presents the background information regarding Gnutella and GnutellaSim. Section 4 presents the overview of our simulations, and section 5 presents the results of Gnutella simulations. Finally, section 6 concludes the paper.

## 2 Related Work

The effect of perturbation is studied in a different context but not studied yet in the overlay networks. Birman *et al.*[9] studied the effect of perturbation in the context of multicast protocol. In their simulation, virtually synchronized multicast groups were used to study the effect of perturbation. They measured throughput of a live node in the presence of “perturbation” - some fraction of the multicast group members put to sleep for some percentage of each second. Their result shows that even with a single perturbed group member, the throughput drops rapidly and the throughput drops more rapidly as the size of the group increases. Even though we measure the success rate rather than the throughput in our simulations, we can see similar results with this study. The success rate drops rapidly with small fraction of perturbed nodes.

Many studies have shown that the arrival rate and the departure rate of nodes in peer-to-peer systems are very high, which proves the instability of peer-to-peer systems. Bhagwan *et al.*[4] showed the availability of Overnet peers. Users in Overnet are identified by immutable IDs which can eliminate the incorrect use of IP as an identifier. They implemented a crawler and a prober to measure the availability. The crawler is used to discover peers in the Overnet network, and the prober is used to probe alive peers so that the availability can be measured. They found that large fraction of hosts join and leave per day. The observed arrival rate and the departure rate were almost same.

S. Sarouj *et al.*[10] studied node availability of Napster and Gnutella. Their methodology is discovering peers in the network first, and probing each of them afterwards. For probing, they tried to open a TCP connection to each node. Their result can be summarized as the best 20% of Napster peers have an uptime of 83% and more, and the best 20% of Gnutella peers have an uptime of 45% or more.

## 3 Background

In this section, we present an overview of Gnutella, and GnutellaSim [5].

### 3.1 Gnutella

Gnutella[6] is an unstructured peer-to-peer file sharing system. Gnutella protocol has 6 messages, *Connect, Ping, Pong, Query, Queryhit*, and *Push*. *Connect* message is a request for connection. Each node sends this message to maintain a link between itself and its peers. *Ping* is a maintenance message to discover new peers. *Pong* is a response to the *Ping* message. Whenever a node receives a pong to the ping that it sent, it records the sender of the pong and tries to connect to the sender. A *query* is a request for an object in the network. It is flooded into the overlay and any node who
can reply to this query sends QueryHit message back to the sender QueryHit is backrouted to the original sender following the reverse path that the matching query was routed through. Push is a message to receive a file from a peer behind a firewall or a NAT box.

3.2 GnutellaSim

GnutellaSim[5] is an extension to ns-2 simulator. Gnutella protocol is implemented as an application on top of ns-2. Routing substrate is separated from the file sharing application, so that various simulations can be easily done by the simulator.

4 Gnutella simulations

The simulations measure successful replies to queries sent. All simulations are conducted with 100 nodes, and one object is replicated and stored in 5 nodes in the system. A query for that object is flooded into the overlay network and records the number of replies. There are 5 senders in the system. The reason that we choose multiple senders instead of just one sender is that Gnutella network is dynamic, and it’s very hard to ensure that the topology is connected all the time. Thus, to minimize the effect caused by a partitioned overlay, we choose multiple senders that can be spread among possible partitions.

The total simulation time is 2000 seconds in ns time. Every node starts in between 0 to 1000 seconds randomly, and does nothing but bootstrapping to give some time for constructing an initial topology. For bootstrapping, every peer contacts a bootstrap server to participate in the overlay. Then, the bootstrap server picks random nodes that are available at that time, and returns a list of those nodes to the new peer (The bootstrap server can return at most 20 available peers in our simulations). From 1000, each node starts either sending queries or flapping.

Since the seed for the random number generator is the same, the initial topology is always the same for every simulation. Figure 1 shows the initial topology extracted from a simulation in which every node stays in the overlay network all the time, i.e. the flap rate (see the explanation below) is 0. Since the topology is extracted from the simulation, it doesn’t contain every node. It only shows the nodes which receive a request from one of the 5 senders. The reachability is limited by TTL value, which is 8 in our simulations.

There are six variables in our simulations;

- Flap rate(r) : This is the probability of going offline for each node. At the beginning of every period, each node decides whether to go offline or online based on this probability.
- Idle period(i) : This is the online period. Since every node except the query-sender doesn’t query anything, the period is idle period.
- Offline period(o) : This is the offline period. After going offline, the node waits o clocks and decides whether to go online or stay offline based on r.
- Query period(q) : This is the period for the query-sender. The sender sends a query and waits q clocks. Note that this is the only parameter that affects the behavior of the sender. Other parameters are only for the nodes other than the sender. In our simulations, the query period is fixed at 100.
- Ping interval : Gnutella uses ping-pong to maintain the peer links of each node.
• Connection timeout: This is the connection request timeout value. In our simulation, the
timeout is fixed at 60 seconds.

To differentiate node flaps caused by user behaviors from intermittent network congestions, we
simulate two scenarios.

**No Join on Return (NJOR)** First one is “No Join on Return (NJOR)”. In NJOR, flapping nodes
do nothing when they go offline. No cleanup procedure is done at all, which means that they
keep the list of peers, and they do not report their departures to the bootstrap server. Also,
when they come back online, they do not initiate the bootstrap protocol, but run normally
from the state with which it went offline.

**Join on Return (JOR)** In contrast to NJOR, peers do every normal procedure in “Join on Re-
turn (JOR)”. They clean their states up and report the departure to the bootstrap server.
They also initiate the bootstrap protocol when they come back online.

GnutellaSim requires a topology on which an overlay network can be constructed. We use 105-
node transit-stub topology generated using GT-ITM[8]. This topology contains 1 transit domains,
which contain 5 transit nodes. Each transit node has 4 stub domains on average and each stub
domain contains 5 nodes on average. Consequently, there are 5 transit nodes and 100 stub nodes
in the topology. 5 transit nodes are connected with each other with probability of 0.6, and stub
nodes are connected with each other with probability 0.42. Gnutella nodes are only attached to
stub nodes.

Our simulations reflect the power-law property of Gnutella network as shown in previous
studies[1] [2] by having the maximum degree of each node follow the power-law distribution. We
choose 2.3 as the parameter $k$ in power-law graphs since [2] shows that the 2.3 is the parameter for the real Gnutella network. However, the minimum value of the maximum degree is limited to 3 since we consider that degree 1 is too conservative for the simulations.

5 Result

Figure 4 shows the effect of flapping without ping, but with JOR. As the offline period goes longer, the number of replies decreases. But surprisingly, the number of replies increases for shorter offline periods. This is the effect of JOR as we can see if we compare figure 4 to figure 5 (we can compare directly because the numbers of replies are the same with the flap rate of 0 in both plots). JOR actually acts as a peer discovery algorithm, which is better than the ping-pong mechanism since each node reports its departure and arrival to the bootstrap server as well as it gets a list of available peers from the bootstrap server when it joins the overlay. It obviously takes less time than the ping-pong mechanism. Thus, with the flap rate of 90% and the offline period of 1 second, the overlay network is refreshed effectively every minute. To get a sense of how the topology changes, figure 2 and 3 shows two traces of replies taken from JOR simulations with flapping rate 0.3 and 0.6 between 1400 and 1460 time range. Idle period of 54 seconds and offline period of 6 seconds are used in both traces. Two traces show a completely different set of request-reply paths among various flapping rates. In figure 1, node 87 is not reachable from node 85 because the number of hops exceeds the TTL value but in figure 2, it is reachable from node 85. Likewise, node 39 also can reach node 87, and node 7 is also reachable from node 85.

Figure 6 and 7 show the effect of ping on both JOR and NJOR. In figure 6, ping intervals of 60, 90, and 120 are used. Ping intervals of 15, 30, 60, 90, and 120 are used in figure 7. Both the
idle period and offline period are fixed at 30 and the flap rate is fixed at 30%. The reason that we don’t present the results of ping intervals of 15 and 30 in figure 6 is worth mentioning here. GnutellaSim drops packets if a buffer of a socket is full when forwarding packets to other peers. With 15 and 30 ping intervals, too many packets are dropped due to the limited buffer. Even though we use the maximum value for buffers, we can’t eliminate packet drops when ping intervals are 15 and 30. Since the effect of congestion is not a subject of this paper, and it deserves a whole set of simulations and analysis, we don’t present the results of congestion. However, even these incomplete simulations suggest that short ping intervals may prevent the overlay from functioning normally.

Both plots show that the number of replies increases as the ping interval goes shorter. Especially, JOR with ping shows the best performance even with the ping interval of 120 seconds, which is the default setting in GnutellaSim. However, NJOR with ping shows an unstable result. To further look into this behavior, figure 8 shows the number of replies over time. Only the ping interval of 15 seconds outperforms the simulation without ping, which suggests that ping-pong can help increasing the performance only when the ping interval is shorter than the offline period. However,
the stiff curves observed in ping intervals 60 and 120 suggest that certain topology changes can bring the file replicas close to the senders.

6 Conclusion

We study the effects of perturbations on Gnutella network using GnutellaSim. A small percentage of perturbed nodes helps increase the overall number of replies when each node contacts the bootstrap server whenever it comes back online. This is because the bootstrap server maintains available peers and gives a list of available peers to a new peer. However, if flapping nodes do not contact the bootstrap server when they come back online, the number of replies decreases continuously as the flap rate gets higher. Ping-pong mechanism of Gnutella helps increase the number of replies especially when it is used with JOR. However, when it is used with NJOR, the number of replies increases only with the ping interval shorter than the offline period.
Figure 8: Number of replies over time with the flap rate of 30%. Idle period is 30 and offline period is 30. No Join on Return.

References


