I Jumped in the GnutellaNet and what did I see ? Lessons from a simple Gnutella network simulation

Thomas Miconi
thomas.miconi@free.fr

October 4, 2002

Abstract

This paper discusses various aspects of Gnutella-like networks (more precisely, networks of Gnutella Ultrapeers) and of their structure. We compare Gnutella networks with tree-structured networks and expose their differences, describing some nasty effects that lead to severe bandwidth waste in Gnutella networks. We support these ideas with a very simple simulation of a Gnutella network and its behaviour with one single request. From this experiment we conclude that while tree-structured networks are both fragile and slow to scan through, having as few outbound connections as possible for every host (ideally 2) is the best behaviour from the network's viewpoint. We introduce a possible mechanism, "idle connections", for enhancing the stability of the network without increasing the number of connections.

1 Structural problems of the Gnutella network

1.1 Tree structures vs Gnutella networks

In terms of number of connections, tree networks are the most efficient way to connect a set of nodes together. With a tree architecture, connecting N nodes together requires N-1 connections, regardless of the way the tree is organized. If a network of N hosts has M connections, and if M is greater than N-1, then the $(M - (N - 1))$ "excess" connections are called redundant or "loop" connections, because they lead to emergence of loops : a loop means that there can be more than one path between two given hosts in the network.

If a broadcast request reaches all N hosts within a network, then the minimal number of connections used is N-1. Thus, in theory, the tree structure ensures minimal bandwidth use for broadcast requests.

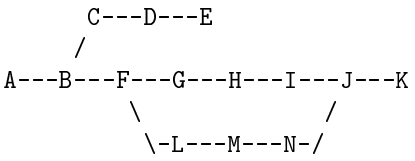
The Gnutella network is the contrary of a tree structure. It is massively redundant, and uses uncountable loop connections - that is, the number of connections is several times higher than the number of hosts, even if you only consider Ultrapeer nodes (as we do in this paper). In the Gnutella network, the number of connections M is several times higher than the number of hosts N, because each host connects to several other hosts when it enters the network.

Loop connections are definitely useful, as is explained below. However, they are also very costly. They are costly not only because they are intrinsically unnecessary, but also because if a request reaches is sent throughout the whole network, then all loop connections will be used twice. This means that a broadcast request in an (N, M) network will use $2 * (M - (N - 1))$ connections for absolutely nothing. For a 10000-nodes networks with 50000 connections, this means that there will be about 130000 transmissions - 50000 useful ones and 80000 unnecessary ones.

1.2 The "crossing letters" effect

The fact that loop connections are used twice comes from a "crossing letters" effect. Imagine a network

with exactly 1 loop connection:



We can see that there is a loop connection somewhere in the [F-J] section. Of course none of them can be singled out as "the" loop connection, but suppressing any given connection in this section would bring us back to a pure tree structure.

Imagine a request is initiated by host A and broadcasted through the network. It will reach hosts B, C, D, E and F without any surprise. But then it enters the "loopy" section in F.

Let us assume for a while that all links are equivalent in terms of transmission speed. The request will reach G and L almost simultaneously. Same thing for H and M, I and N. But what happens between I, N and J ?

I and N try to pass their request on to J. Obviously I and N cannot transmit their request exactly in the same clock cycle, and even in that case J cannot treat them simultaneously (unless it has more than one processor). So one of the two requests will arrive before the other. Let us assume that it is the request coming from I. As soon as J receives this request, it will try to pass it on to all the other nodes it is connected to, that is, K and N. But the connection between J and N is already being used by N to transmit the very request that J is sending back!

If we relax the assumption that transmission times between all hosts are almost equal, we only push the problem backwards. For example, if the connection between F and L is significantly slower, then the request will reach host J before it reaches host N, N will receive the request from J before it receives it from M, and thus the crossing letters effect will happen somewhere between hosts F and M (included).

There is a simple way to alter this effect : if every host waits for a given time before sending out any request, then it will be able to intercept identical requests coming from different hosts during that time, and not send it back unnecessarily. However, for this method to bring a real advantage, the delay between reception and transmission must be significantly superior to transmission time between two hosts. Even then, it is still perfectly possible that a request is sent by a given host A to another host B while B already sent this request to A, but the request is still "in the tube". In the current model of the gnutella network, where no such measure is taken, the crossing letters effect happens as many times as there are loop connections in the network - hence the unnecessary transmissions described above.

1.3 Practical questions

While these unnecessary transmissions lead to significant bandwidth waste, it is obvious that the theoretical solution of a tree-structured network is unfeasible in practice. There are at least two reasons for this:

1- Trees are extremely fragile. Since there can be only one path between any two hosts, if you break one connection, you split the whole network apart. People who have used IRC are all too familiar with this "split" phenomenon, which derives from the tree structure of IRC networks.

2- To scan a whole tree, you need a very high TTL, so the request may take a long time to reach all hosts. At each time step, a Gnutella request is likely to reach more hosts than the same request in a tree network. In this regard, loop connections are seen as shortcuts through the network.

So the question is: what is the most sensible behviour in terms of inbound/outbound connections for any given host ? To answer this question, we must remember that the total number of connections within a network of N hosts is equal to N multiplied by the number of hosts to which a new host tries to connect when it enters the network, that is, to the number of "outbound" connections for every host. If all hosts have K outbound connections (i.e. they connect to K hosts when they enter the network) then there are N * K connections within the network (possibly $(N-1)*K$ depending on how you connect the first host in the network). K = 1 brings us back to a tree structure.

It is important to stress the difference between inbound and outbound connections: allowing more

people to connect to you (that is, having more inbound connections) will not increase the total number of connections within the network: the people who connect to you would have connected to someone else anyway. The number of hosts to which every new host connects at startup (that is, the number of outbound connections) is the really important factor: it directly determines the total number of connections within the network, and thus the bandwidth used (and wasted).

Of course, this only applies at the network level. When it comes to an individual host, the proportion of inbound and outbound connections has little importance: since relaying is non-directional (inbound and outbound connections are treated the same way) the amount of bandwidth consumed is determined by the total number of connections.

So the question becomes: for a fixed total amount of connections, what is the most sensible mixture of inbound and outbound connections, that is, how many hosts should you connect to at first, and how many connections should you leave vacant for other hosts to connect to ?

2 Experiments

2.1 Experimental settings

Our experiments were led with a very simple model of a Gnutella-like network. As explained above, we only consider Ultrapeers in this study. The network was created in the following way: first, a small number of hosts were connected together in one single line (a "flat tree", with only one branch), thus creating the seed of the network. Then all hosts were connected to the network by establishing K outbound connections with other, already connected hosts. Even the hosts of the seed went through this process in order to ensure that all hosts had exactly K outbound connections. The resulting network seems to be a good approximation at a Gnutella network at any given time.

All hosts had the same maximum number of connections. If a host tried to connect to another host whose connections were all taken, it had to retry with another host.

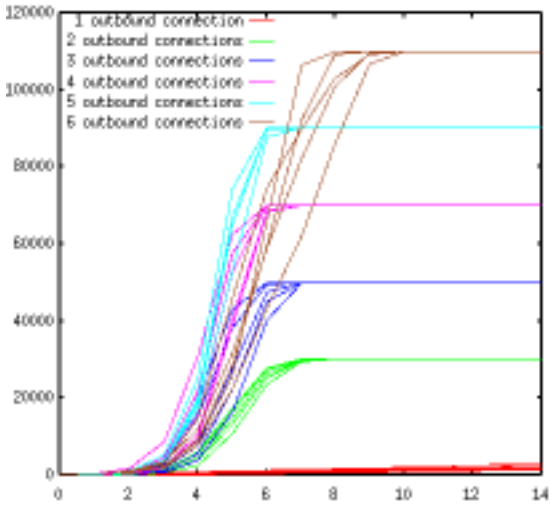


Figure 1: Number of connections used after each hop. These results correspond to the expected behaviour.

Request transmission was simulated in the following way: a host is randomly chosen to be the initiator of the request. It transmits the request to all hosts that are connected to it (inwards or outwards, of course). Then every contacted host transmits the request to all other hosts connected to it, except the one from which they got the request, and the request's TTL is decreased. This process is repeated until the request's TTL reaches 0.

We used a network of 10000 hosts, with a maximum of 13 connections each. The seed was composed of 100 hosts for every run. The initial TTL for a request was 14 (this ensured that all hosts were reach, except for the tree-structured network). The number of outbound connections K varied from 1 (tree-structure) to 6. For each value of K, we ran 5 experiments.

2.2 Results

Figure 1 shows the number of hosts reach after each hop. Most curves are strongly intricated, except for 1 and (more interestingly) for 6 outbound connections. Figure 2 shows the number of connections used after each hop. Finally, figure 3 gives the most interesting

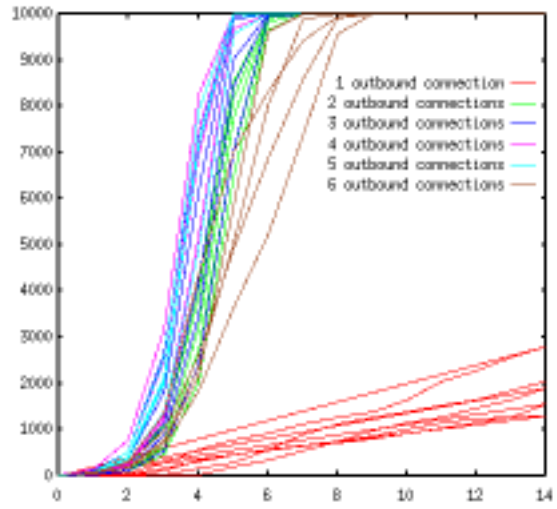


Figure 2: Number of hosts reach after each hop. Notice that with K=6, the curves are significantly lower than with other values. Besides, the tree structure (K=1) only covers a small part of the network.

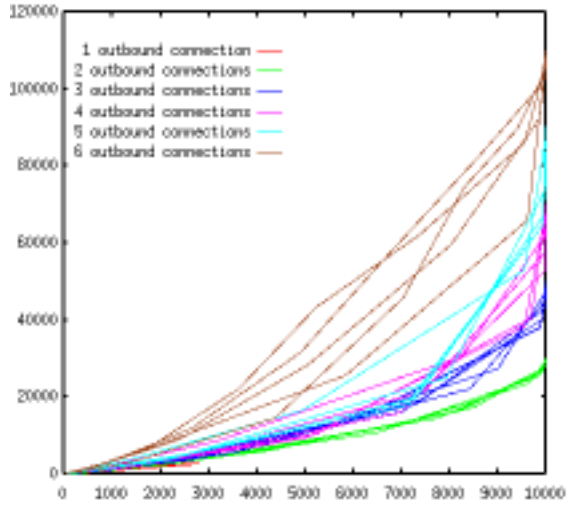


Figure 3: Number of connections used in regard to the number of hosts reach. This curves indicates the efficiency of the network. Since the tree structure (K=1) could only reach a small number of hosts, its curve is very short.

results by showing the relation between the number of hosts reach, and the number of connections used to reach these hosts: it is the real efficiency curve of the network.

The first thing to notice in these tables is that whatever the circumstances, you can always reach more hosts using fewer connections (and hence fewer packets) by reducing the number of outbound connections. This is even more true if you want to reach the whole network : as expected, the number of connections used is about $2*(M-(N-1))$, with $M = K*N$ being the total number of connections within the network. But this rule also holds if you only want to reach a part of the network, as the curves indicate.

However, things are not that simple: first, the difference in efficiency may take some time to become significant, as figure 3 indicates: except with 2 outbound connections, most curves are difficult to sort out if less than 7000 hosts are reach. Second, bandwidth consumption is not the only parameter: if you reduce the number of outbound connections, it takes fewer connections to reach an equivalent number of hosts, but intuition indicates that it may also take a higher TTL to reach the same number of hosts, which means more time taken.

Indeed, we see that with K = 2, you reach more or less the same amount of hosts at hop 6 as with K = 4 at hop 5. However, this time penalty must be put in regard with the fact that half as many connections were consumed !

Moreover, an interesting effect can be seen in figure 1: we can see that above a given threshold, increasing the number of outbound connections actually decreases the speed of propagation. With K = 6, you need much more hops (i.e. a higher TTL) to reach the same number of hosts than with K = 5, 4 or even 2 !

The idea that having more connections within a network might handicap the propagation speed may seem a little bit counterintuitive. In fact it is easy to understand if you realize that the network is built in an aggregative way, with new hosts joining the network one after another. Imagine that you just entered the network. If you use many outbound connections, you are using up some inbound connections from the network, and thus it will be more difficult

for other newcomers to connect directly to the heart of the network - some of them will have to connect to you instead.

So by using many connections, you added one host (you) between them and the network, and this makes the network less dense. In other words, because of the limited maximum number of connections for every individual host, we end up with the somewhat puzzling result that having too many outbound connections may, in some cases, reduce density.

We have seen that having fewer outbound connections makes the network more efficient (fewer packets to reach more hosts), and that this effect becomes eally important with K = 2. However, it is obvious that having only 2 outbound connections can make the network somewhat fragile. An IRC-like "split" is very unlikely because, even if you lose your two outbound connections, the hosts that connected to you can still provide a link to the rest of the network. However, this can still degrade performance seriously, both for you and for the network.

3 Idle connections

Is there a way to enhance the security of the network and of every host without increasing the number of outbound connections (and the network load) ? We propose a simple scheme that may help bringing more security by allowing for a shorter replacement time for any broken outbound connection, allowing for all hosts to reconnect almost instantaneously in most cases. This mechanism is based on "idle" connections.

Idle connections, as their name indicates, carry no kind of data. They are simply links between networks, such as idle TCP connections, and their sole purpose is to say "Hello, I'm here, and if you ever want to connect to me, I'm ready". If the host in question isn't ready any more (e.g. all connections taken), then the idle link is simply closed, and the initiating host might have to find another friendly host.

Let us explain this better: when a new host enters the network, it establishes K regular outbound connections. But it also establishes a given number of idle connections with other hosts that still accept connections. Thus, if one of its K outbound connections gets broken, it simply turns an idle connection into a real one, which restores its full connectivity in no time.

All hosts that can still accept connections should also accept idle connections. As soon as the last connection is taken, all idle connections are closed, and thus the hosts that initiated these idle connections know that they'll have to find another partner. For more security, all hosts should establish several idle connections, so that if one is closed, this doesn't make them too vulnerable.

This system may enhance security, but it does have drawbacks. Idle links management may consume CPU and even have some impact on network load (because of management packets). Obviously this subject calls for more discussion.

4 Conclusion

The conclusion of this paper is very simple and holds in three points:

1. The most efficient structure in terms of bandwidth use is the tree structure. Since the Gnutella network rejects the idea of a tree structure (and rightly so, since a tree is extremely fragile and takes forever to search), it is bound to consume more bandwidth, but the crossing letters effect makes this much worse (twice worse, actually).

2. Allowing for one more host to connect to you is almost always better than connecting to one more host yourself. You don't loose anything in terms of connectivity, and the network as a whole works better. Connect to 2 or 3 reliable hosts, and let as many people as possible connect to you !

3. The idea of idle connections might be explored in order to enhance the stability of the network without increasing the number of outbound connections. A stable network in which all hosts have only 2 outbound connections could be searched very efficiently.

5 Acknowledgements

The title of this paper was shamelessly stolen from a Radiohead song. Moreover, this work benefited greatly from the positive influence of Chico Buarque's music. We encourage the reader to consult "Essa moca ta diferente" or "O que sera" for more information.