small networks and can be used to check simple properties of the Gnutella network.

1 Background and Motivation

The Gnutella protocol allows decentralized peer to peer file sharing[1]. The Ping/Pong section of the protocol allows servants to increase the number of connections they have to other servants. (Each computer in the Gnutella network is called a servant, because it is both a client and a server.) A servant wants to increase the number of its connections to avoid being cut off from the network. When a servant initially connects to the network, it is connected to only one other servant; if that servant dies, the new servant is completely cut off from the network. In addition, over time, the connections a servant has may become stale as servants leave the network, and new servants enter.

We hoped to create a model that would give us insight into the Ping/Pong protocol, and allow us to check interesting properties of the protocol. Although limitations in Alloy prevented us from reaching this goal, we were able to check some assertions for small networks.

2 Summary

Our model looks at the protocol at the end to end layer rather than the transport layer; we do not deal with TCP packets or messages sent between servants, but rather with Gnutella messages that could be sent to more than one servant. Each servant in the model is connected to one or more other servants. Servants send and receive Messages in order to transfer information about other servants in the network. A Message is either a Ping or a Pong. Every message has a Globally Unique ID (GUID) to identify the request it is participating in on the network. In addition, each Message has a time to live (TTL) field signifying how many hops the Message has until it will be dropped. In the Gnutella protocol, a GUID is a 32 bit integer tagging the Ping message that originated all other Ping/Pong messages in the network. In our model, a GUID is simply a pointer to the Ping message that originated the request. We do not use the information in the GUID in any way; it was simply convienient to use this as the GUID rather than creating an additionaly type.

When a servant receives a Ping, it creates a new Ping with the same GUID and a TTL of one less than the original Ping. It forwards this Ping to all of its connections. It also adds an entry to

---

[1] All information about the Gnutella protocol is taken from the documents at http://rfc-gnutella.sourceforge.net/

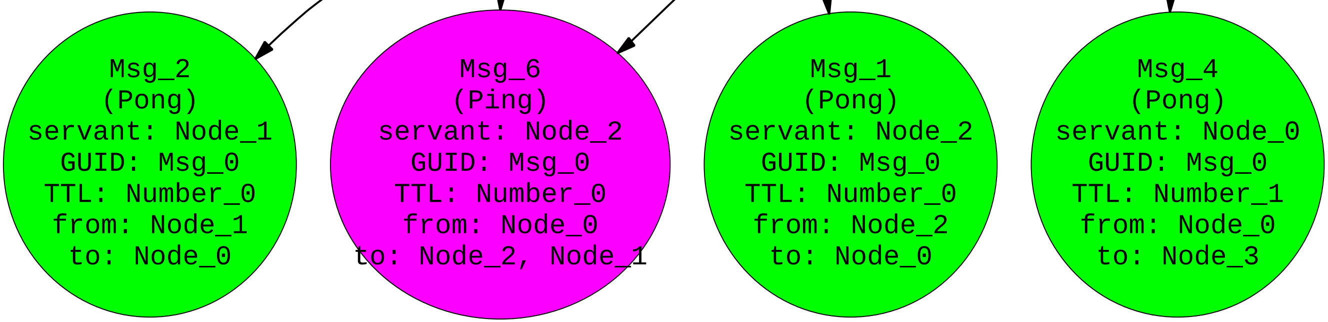

Figure 1: An sample visualization from our model.

its routing table mapping the GUID of that Ping to the servant from which it received the Ping. This entry enables it to route future Pongs down the correct path to the servant that sent the original Ping. In addition, it sends a Pong message containing its connection information to the servant that originally sent the Ping. (In Gnutella, it would send a Pong containing its IP and port; in Alloy, it sends a pointer to itself.)

When a servant receives a Pong, it looks in its routing table for an entry mapping the GUID of that Pong to another servant. It forwards the Pong to that servant. If it is not already connected to the servant that originally sent the Pong, it adds a connection to that servant.

Figure 2 shows a sample visulization from our model. It shows a four servant network in "Y" configuration at a state midway through the Ping/Pong protocol. Node_3 originally sent a Ping (Msg_0) to Node_0. At this state, Node_1 and Node_2 received a Ping (Msg_6) forwarded from Node_3 for Node_0 and have just replied with a Pong (Msg_1 and Msg_2). In addition, Node_3 just received a Pong (Msg_4) from Node_0 in response to its Ping. Node_0 has added an entry mapping Msg_0-¿Node_3 to its routing table, so that if it receives a Pong with a GUID of Msg_0, it knows to send it to Node_3, rather than to another one of its connections.

3 Evaluation

In order to test the properties of our model, we checked the connectedness properties of a servant after it sent an initial Ping. We tried asserting that the servant would be fully connected after the contains few interesting assertions that can be checked in reasonable time for a reasonable number of servants. Although we were able to create a model that sent the right messages at the right times for small networks, it took a prohibitively long time to run on networks of only four nodes.

We had to make many optimizations in order to make our assertions run in reasonable time. For example, we altered the messaging module in order to eliminate the parts we were not using to reduce the number of atoms in the model. In addition, we eliminated many of the fields in our visualization, and used functions to compute the properties instead. For example, we originally had a relation representing new connections added as the result of a Pong message. We added a few simple rules to populate this relation and it worked well for networks of three servants. This relation was used in order to see if a servant was fully connected after the Ping/Pong protocol completed. However, in order to make four servant networks run in reasonable time, we had to eliminate this relation, and write a function to compute it instead. Although it was much faster, we could no longer see new connections in our visualization.

We had to create a plethora of facts to prevent Alloy from sending spurious messages and inserting spurious entries in the routing table. For example, after telling Alloy to add the correct rows to the routing table of a node after forwarding a Ping, it added the correct row, but also filled the routing table with spurious rows. We had the same problem when telling nodes what messages to send.

5 Conclusion

We created an Alloy model of the Gnutella Ping/Pong scheme. Our model allowed us to visualize the protocol for small networks and explore simple assertions, but we were unable to scale Alloy to explore interesting assertions in larger networks.