**Gnutella Crawler**

**Programmed By: Zachary Paladino**

**Date: 02/16/2009**

**Code Structure:**

My Gnutella Crawler was programmed in C++ using Visual Studio 2008. The input for the crawler is from the command prompt. The input has to be in the form of host[:port][/path] (where the parts in [] are optional) and then the number of threads and how many peers you want to connect too are the next arguments. If the number of threads and peers are not specified then they are defaulted to one. The Crawler is broken down into a few key functions. After the initial parsing of the values given by the user the host, port, and path of the Gnutella repository passed into winsock\_test where a TCP HTTP connection is set up and the response from the server is then parsed and the peers are put into a Breadth First Search (BFS) Queue. Once the peers have been found the number of threads that the user gives are started and the Gnutella Crawl starts. Every thread takes a peer from the BFS queue and connects to it through the GNU\_winsock\_test function. Once the connection has been established the response is parsed and the peers and leaves are returned back to the thread. The new peers are put back into the BFS Queue and both the peers and the leaves are put into the overall set of peers. This process continues until the number of connections that was established by the user has been reached. The data that has been gathered by the crawl is then put through a DNS lookup process to fins the hostnames of the peers. The DNS lookup starts as many threads as the initial crawl started, which is specified by the user. Once all of hostnames have been found then they are parsed to find their country codes and their main hostname. This data is then stored into files to be sifted through later.

**The Experiment:**

The crawl that I experimented with used 5,000 threads to connect to 200,000 peers. My computer which has a Intel quad core processor and 2GB of ram using Windows 7 professional edition ran the experiment in 555,493ms or around nine minutes. Out of the 200,000 peers that were connected too only 29,159 peers were responsive and found relevant data. The average time to crawl responsive peers were 0.052 ms. There were 371,060 peers and 210,677 leaves were found in the crawl. I also experimented with 1,000 threads for 200,000 peers and it was less efficient than 5,000 threads. It took around twenty minutes to crawl the same amount of peers that the 5,000 thread experiment crawled. That was around double the time for the same type of data to be gathered. Which logically proves that the more threads that are involved the quicker the crawl will finish. The data from the 5,000 thread experiment were all part of the DNS lookup for the data that follows. The hostnames that were found had a good amount of diversity but the six shown in Figure 1(a) of Appendix A were the most prevalent. These six were Comcast.net with 23,364 peers, RR.com with 14,638 peers, Verizon.net with 8,172 peers, Cox.net with 6,364 peers, Sbcglobal.net with 6,100 peers, and Optonline.net with 5,500 peers. The second set of data that was gathered from the crawl was the diversity of country codes. There was a wide variety of country codes that were found but the top six that were the most prominent were, .net, .com, Canada (.ca), Brazil (.br), Mexico (.mx), and France (.fr). The final statistic that was derived from the crawl was the diversity of user agents of the peers. Figure 1(c) shows the different versions of user agents that were found during the crawl. The statistics found that LimeWire version 5.4.6 was the most prominent followed by LimeWire version 5.3.6. The rest of the user agents found were a mixture of LimeWire and other agents such as Bearshare.

**Conclusion:**

The Gnutella Crawler that I created is a powerful tool to look up a variety of statistics on a Gnutella network. If I did not test this on a school network I would have received more connections due to the Network Address Translation (NAT) refusing some incoming connections from .edu sources. It is also good practice to have a designated DNS server to run this on so you do not flood one that is not designed for the amount of connections that this program creates.

**Appendix: A**

**Figure 1(a): Diversity of Hostnames**

**Figure 1(b): Diversity of Country Codes**

**Figure 1(c): Diversity of User Agents**