A Prototype Power Management Proxy for Gnutella Peer-to-Peer File Sharing

Miguel Jimeno† and Ken Christensen
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
{mjimeno, christen}@cse.usf.edu

† Miguel Jimeno is on leave from Universidad del Norte, Colombia

Abstract— **In order to be part of a peer-to-peer (P2P) file sharing network a host must be fully powered-on all of the time. In addition to providing a user interface, a P2P host handles query messages and serves requested files. In this paper, we describe the development of a prototype Gnutella-like P2P power management proxy sub-system that handles query messages. This can allow desktop PCs acting as P2P hosts to enter a low-power sleep state for most of the time and be woken-up by the proxy only when needed to serve files. TCP connections with neighbors are maintained by the host when it is awake and by the proxy when the host is sleeping. Experiments show that a low-cost Freescale ColdFire processor can effectively proxy for a P2P host. This suggests that a controller for a Gnutella P2P proxy could be co-located on an Ethernet NIC at low cost. This could lead to significant energy savings by allowing P2P hosts to power manage into a low-power sleep state when not in active use.**

I. Introduction

The growing energy consumption of Information Technology (IT) and Consumer Electronics (CE) equipment is now a major concern. Research into reducing the energy use of IT and CE equipment is of great economic and environmental significance. A typical desktop PC consumes 120 W when fully powered-on [7]. Operating one additional PC fully powered-on all of the time in a typical US residence would add about 10% to the yearly electricity bill [1]. This is a non-trivial impact. Increasingly, networked applications have been found to *induce* energy use in PCs and other networked devices. Induced energy use occurs when devices are required to remain fully powered-on – even when no user is present and network access is only sporadic or incidental – in order to respond to network protocol messages. One example of this is peer-to-peer (P2P) file sharing. A desktop PC, or host, participating in a P2P network must be fully powered-on at all times in order for it to make available its files to other P2P hosts. This is the case even if the actual time during which files are downloaded from a P2P host is very small.

In [4] a power management proxy was designed and evaluated for the UPnP protocol. In [3] we proposed and outlined a direction for a power management proxy for P2P. In this paper, we describe the design, implementation, and preliminary evaluation of a prototype P2P power management proxy for the Gnutella protocol.

This material is based on work supported by the National Science Foundation under Grant No. 0520081.

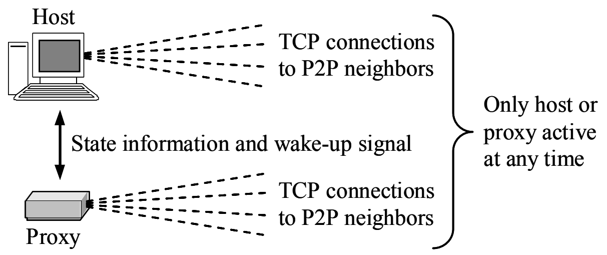
Host
TCP connections to P2P neighbors
State information and wake-up signal
Only host or proxy active at any time
TCP connections to P2P neighbors
Proxy

Figure 1. P2P host with a proxy sub-system

II. Overview of the Gnutella P2P Protocol

A P2P network is an overlay network on the Internet. P2P host neighbors are connected by TCP connections. Neighbors need not be physically nearby, and the process of identifying neighbors involves a bootstrapping process. Gnutella is a popular P2P file sharing protocol that uses a query flooding to find files in the network. The standard protocol version is 0.4 and defines five message types; Ping, Pong, Query, QueryHit, and Push [2]. The Query message is used to find files; the QueryHit message is the response from a queried P2P host that contains a queried-for file. Files are downloaded using HTTP.

III. Design and Implementation of a P2P Proxy

In [3] we proposed that a P2P power management proxy could be designed to operate in a low-power microcontroller. In particular, we considered the effective use of a Bloom filter for look-up of files stored (and shared) within a P2P host. The shared files could not be stored in the proxy due to its limited storage capabilities. When a request for a file – in the form of an HTTP GET – was received at the proxy, the proxy would wake-up the host and the host would then serve the file.

A. Design of a P2P proxy

Figure 1 shows a P2P host with a proxy sub-system. The proxy sub-system can be co-located within the host (e.g., on an Ethernet NIC) or in another device (e.g., a LAN switch). The proxy subsystem takes over for the host when the host enters a low-power sleep state. This would occur when no user is present. For example, this would occur in a desktop PC after a fixed period of inactivity (where inactivity is defined as no keyboard or mouse activity). The goal is that the proxy sub-system would consume much less energy than the host. The

```
Program hostProgram()
  call listen()
  call neighborConnect()
  start queryHandler()
  start getServer()
  while (true)
    prompt user for name of file to search
    generate Query message for file
    send Query message to all neighbors
    receive and display results of QueryHit messages
    prompt user for host name for download
    use HTTP to request file from selected host
```

Figure 2. Main program for host

```
Program proxyProgram()
  call listen()
  call neighborConnect()
  start queryHandler()
  start redirector()
```

Figure 3. Main program for proxy

```
Process listen()
  while (true)
    listen for and accept incoming connections
    update neighbor list with accepted connections

Function neighborConnect()
  connect to all hosts in neighbor list

Process queryHandler()
  while (true)
    for (each connection)
      wait to receive a Query message
      forward Query message to all neighbors
      check for queried file
      if (file exists)
        send a QueryHit message
```

Figure 4. Functions and processes common to host and proxy

```
Process getServer() – Standard HTTP server

Process redirector()
  while (true)
    wait for a connection on port 80
      if (receive an HTTP GET)
        wake-up the proxied host
        send an HTTP 302 to redirect request to host
```

Figure 5. Processes specific to host or proxy

proxy sub-system will be resource constrained and cannot store files shared by the host. If the proxy is a separate sub-system not physically contained within the host, it may have a different IP address from that of the host.

A P2P application in a host includes a main program that supports a user interface for generating Query messages,

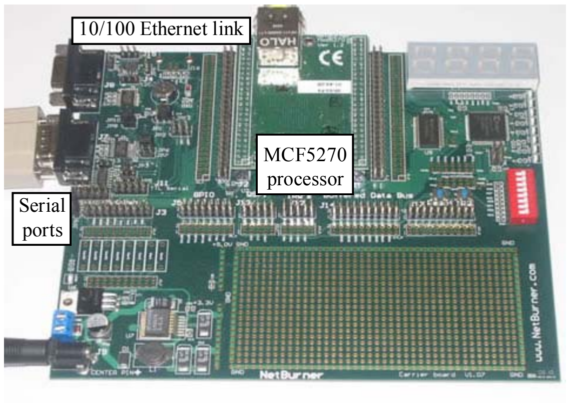

Figure 6. Netburner development kit

displaying query results, and initiating file downloads. The P2P application must also include capabilities for initiating and accepting connections to and from neighbors, receiving and forwarding Query messages, generating a QueryHit message when a Query message for a file that is being shared is received, and serving files that are requested with an HTTP GET. A P2P proxy need only support a subset of this; which are capabilities for initiating and accepting connections to and from neighbors, receiving and forwarding Query messages, generating QueryHit messages, and waking-up the host when an HTTP GET is received. Figure 2 shows the main program for the host and Figure 3 for the proxy. Figure 4 shows the processes and functions common to both the host and proxy. Figure 5 shows the processes specific to only the host (the getServer()) and proxy (the redirector()). In all cases, processes execute in parallel and do not terminate (e.g., as Windows threads) and functions execute and terminate. The redirector() process uses an HTTP 302 redirect message to cause the requesting host to resend its HTTP GET message. The GET can be forced to be resent to the same or another IP address. The redirection gives the host time to wake-up.

State information needs to be shared between the host and proxy. The state information shared includes:

- Power state of the host – fully powered-on or sleeping
- List of names of files shared
- List of IP addresses of neighbor nodes

The file names can be shared between the host and proxy in the form of a Bloom filter. When control is transferred from the host to the proxy, TCP connections to neighbors from the host are terminated and then re-established from the proxy. When control is transferred from proxy to host, the opposite occurs.

B. Implementation of a prototype P2P proxy

The P2P proxy was implemented using a NetBurner MOD5270 Ethernet Development Kit [5] (shown in Figure 6) with the following specifications: 32-bit Freescale ColdFire processor running at 147 MHz, 512 KB of Flash memory, 8 KB Instruction/Data cache and 2 MB of SDRAM. The system runs the uc/OS operating system. The host was a Dell OptiPlex PC with a Pentium 4 at 3.2 GHz with 1 GB RAM running Windows XP. Both systems support 100 Mb/s

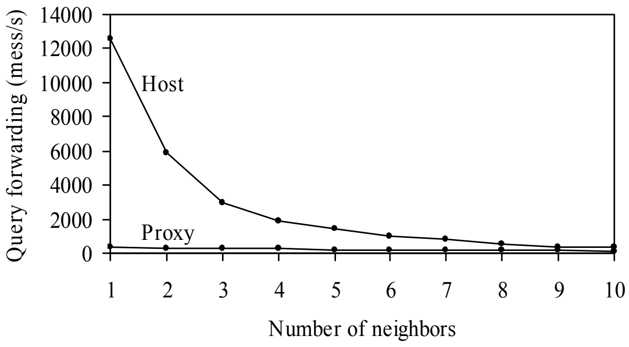

Figure 7. Results from Query forwarding experiment

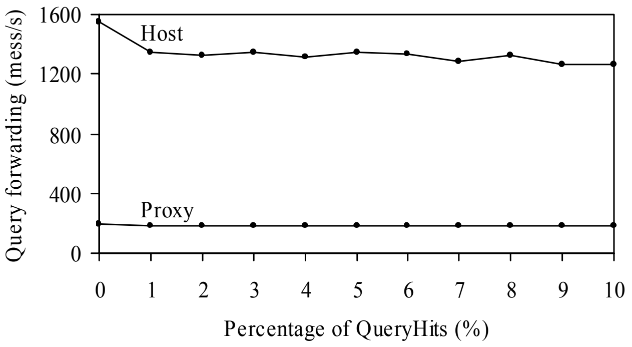

Figure 8. Results from QueryHit experiment

Ethernet. The NetBurner was selected for the low-cost processor it uses and for its ability to be programmed in C.

The uc/OS operating system runs threads as sequential tasks. Thus, the proxy program was implemented as a single task with a main loop where all the processes were run in a sequential manner. In the host, the processes were run in parallel as threads. The proxy implementation used non-blocking sockets that were read using time-outs of one processor tick (which is 100 ns). In the host, blocking sockets could be used in a threaded implementation.

IV. Evaluation of the Prototype P2P Proxy

A key question is, how much processing "horsepower" is needed in the proxy sub-system in order to maintain a reasonable query forwarding rate at all times? The additional overhead time to request and receive a file from a sleeping host also needs to be considered. We designed experiments to evaluate two key measures for our implemented prototype P2P proxy. The two measures were:

- File download time from an awake and sleeping host.
- Query forwarding rate as a function of 1) the number of neighbors, and 2) the percentage of Query messages resulting in a QueryHit message being generated.

For the Query forwarding experiments, a PC running a "Query blaster" was used. The Query blaster was a C program that establishes a connection to a P2P host and sends Query messages as fast as possible. To evaluate the Query forwarding rate as a function of the number of neighbors, we used other PCs to connect to as P2P peers and varied the number of peers from 1 to 10. For this experiment, we generated Query messages for files not in the host, so the proxy would not respond with a QueryHit message. For the second experiment we evaluated the Query forwarding rate where a fixed percentage of the Query messages would result in a QueryHit message being returned. We varied the percentage of Query messages that resulted in QueryHit messages from 0% to 10%.

For the file download time experiment it took less than 1 second to download from an awake host; it took 9 seconds to download from a sleeping host that had to be woken-up. The wake-up time of Windows XP was the dominant factor. Future versions of Windows (Vista) may reduce this wake-up time. The HTTP request did not time out in any case. The results from the Query forwarding experiments are shown in Figure 7. The figure shows the Query forwarding rate per connection. The Query forwarding rate for the proxy varied from 360 to 130 messages per second. The rate for the host varied from 12,547 to 324 messages per second. These results show that as neighbors are added, the query forwarding rate per link was decreased. The results for the QueryHit experiment are shown in Figure 8. Similar to Figure 7, this figure shows the Query forwarding rate per connection. As the percentage of Query messages resulting in a QueryHit was increased from 0% to 10%, the Query forwarding rate remained roughly constant for both the proxy and host. This demonstrates that the overhead to send a QueryHit message is very low.

V. Summary and Future Work

We have designed and developed a prototype power management proxy for a Gnutella-like P2P protocol targeted for a low-cost ColdFire processor. The Query forwarding rates achieved by the proxy were between four to ten times higher than the measured Query message rates in actual P2P networks [6]. Thus, we believe that our proxy can feasibly "take over" for a P2P host. In [3] we calculated an expected energy savings of $38 million per year in the US if 25% of all P2P hosts were to adopt proxy-based power management.

References

[1] Energy Information Administration, "U.S Household Electricity Report," July 2005. URL: http://www.eia.doe.gov/emeu/reps/enduse/er01_us.html.

[2] Gnutella Protocol Specification Version 0.4, 2002. URL: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[3] M. Jimeno, K. Christensen, and A. Roginsky, "A Power Management Proxy with a New Best-of-N Bloom Filter Design to Reduce False Positives," *Proceedings of the International Performance Computing and Communications Conference*, pp. 125-133, April 2007.

[4] J. Klamra, M. Olsson, K. Christensen, and B. Nordman, "Design and Implementation of a Power Management Proxy for Universal Plug and Play," *Proceedings of the Swedish National Computer Networking Workshop (SNCW 2005)*, September 2005.

[5] NetBurner MOD5270 Ethernet Core Module, 2007. URL: http://www.netburner.com.

[6] K. Sripanidkulchai, "The Popularity of Gnutella Queries and its Implications on Scalability," Technical Report, February, 2001. URL: http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html.

[7] US Department of Energy, Energy Efficiency and Renewable Energy, "Estimating Appliance and Home Electronic Energy Use," 2005. URL: http://www.eere.energy.gov/consumer/your_home/appliances/index.cfm/mytopic=10040.