Dynamic Hierarchical Caching for the World Wide Web

Martha H Greenberg
AUP Supervisor: David R. Karger

February 6, 1997

1 The Project

The World Wide Web is a system for distributing documents, including hypertext, images, sound files, plain text, audio, animations and applications. One of the most visible problems for the web today is slow response time due to overloaded networks and servers. Very popular web sites get too many queries at once and respond very slowly or not at all.

A web document is uniquely identified by a Uniform Resource Locator (URL). The URL for a document contains the protocol that should be used to ask for the document, the name of the machine which has the document (the content server), and the pathname by which the machine can identify the document. In the simplest web scheme, a client will directly request the document from the content server every time the user requests to see the document.

Caching is the storage of retrieved documents locally for further use. Therefore, it is not necessary to connect to the content server the next time that document is requested. This technique reduces the number of requests the content server must process. A proxy server is a machine that receives requests from a number of machines; it then either serves the request itself, denies the request or forwards the request to the real server. A proxy cache server is a proxy server that caches the information that went through the proxy. Since the proxy server is usually closer (in terms of network throughput) and less loaded than the content server, the client sees a faster response to their query. Caching also saves network bandwidth to the content server.

In current hierarchical caching schemes, the hierarchy of machines is fixed. Neighbors are other caches which are on the same level of the hierarchy. Parents are on a higher level, and children on a lower level. A cache will make a request to another cache, usually its parent, if it does not have a valid copy of the document being requested. If no other caches have the document, it will request the document from the content server. Machines higher in the hierarchy are responsible indirectly for more clients and will be asked for more distinct documents.

In the Harvest/Squid scheme, each machine knows a list of neighbors and parents to whom it can make requests. The network administrator must give the cache a list of what other caches it can ask, and whether they are neighbors or parents. This list is the same for all documents, so if the cache receives multiple requests for different documents that it does not have, it asks the same set of caches for all of them.

A major problem with current hierarchical caching schemes is that caches near the top of the hierarchy are responsible for a very large number of distinct documents and may become overloaded processing queries for those documents. The ultimate goal of this project is to set up a system in which it is very unlikely for a single caching server to become overloaded with queries relative to other caching servers.

I have been working for David R. Karger in the Algorithms groups of the Theory of Computation Group of the Laboratory for Computer Science. He has designed an algorithm that may solve this problem. My project was intended to determine if this algorithm would work in the current Internet, and if it would be more efficient than the schemes that are currently in use.

The proposed new algorithm differs in that there is a unique hierarchy of parents for each URL. The cache builds this hierarchy in a number of steps. First, an abstract tree of degree d is constructed. This tree and the degree d depend on only on the number of caching servers. Then a hashing function maps `(tree node, URL)` pairs to caching servers. When the cache gets a request for a document, it picks a random leaf in the abstract tree, computes the path from the leaf to the root, and then maps the abstract tree nodes onto actual caching servers using the URL. The cost to the new algorithm is that the cache must have a near-recent list of all the proxy servers in the cache system. This is a benefit, since no caching server is at the top of any fixed hierarchy, all caching servers will be responsible for about the same number of distinct documents and queries.

I have implemented this algorithm by modifying the Squid caching and proxying system from the National Laboratory for Applied Network Research (developed from the Harvest project at the University of Colorado). In order to do this, I have also extended the currently used cache protocol to allow the machines to communicate which machines on the network are using this algorithm. They also have to pass along where they are in the hierarchy for a particular document, so that the next machine does not have to recompute the entire hierarchy.

I have implemented this algorithm for determining what cache to ask for a particular document. I have also integrated it with Squid so that Squid uses this algorithm for determining what other caches to ask for a document, rather than using its fixed hierarchy.

2 Code Overview

All changes are against Squid version 1.1.3. All the changes to the Squid code are set off by the comments (DH stands for Dynamic Hierarchy):

```
/* begin DH changes */
/* end DH changes */
```

When Squid starts up, if it is run with the -d option, the variable opt_dynamic_hierarchy get set. The rest of the changes are conditional on that variable.

At the startup, Squid gets an initial list of all the hosts in network from another server. Right now, only one server can be specified.

Squid then goes through its primary select loop. In this loop, it checks how long it has been since the last update. If it is longer than the set time, it gets a list of all the changes since the last update.

When Squid gets a request that is not in cache, Squid is told that there is only one neighbor, a parent. Instead of getting this parent from the configuration file, as would be normally done, it calls `dhGetParent()`. This procedure builds a tree, maps the URL against the unit interval and finds the correct parent for this machine in that tree. This is done in proto.c.

3 Netserv

Netserv uses a simple TCP based protocol to let hosts participate in the network. The client makes a TCP connection to the netserv server on the assigned port (port 8999 was used for this implementation), and sends a request. The possible requests are: join [⟨points⟩], dele, list, since ⟨time⟩, and help. Join adds the host to the network, dele deletes it from the network, list lists all the hosts in the network, since returns all the changes since the specified time and help prints a help message. Join takes an optional argument of the number of points on the unit interval that the host wants to take. Since takes a mandatory argument of a time in seconds since 00:00:00 UTC, January 1, 1970 (the Unix epoch).

The server responds with a three-digit response code followed by a human-readable explanation. In the cases of list, since and help, it also is followed by the requested information. A response code of the form 1xx indicates a usage message, 2xx indicates a successful request and 4xx indicates a failed request.

4 Modified files from the Squid distribution

src/squid.h
src/main.c
src/proto.c
src/Makefile.in
squid.h - add the global variables and include files needed for the new functions.
main.c - add a flag (-d) to Squid to use the dynamic hierarchy code. get the list of hosts in the network, and keep it updated.
proto.c - change the code to call the dynamic functions when necessary, rather than the Squid ones. tell Squid there is only a single parent, and generate that.
Makefile.in - Add the new object files to the list of ones needed to generate the Squid binary and build the netserv program.

5 Files added to Squid

chash.h
chash.c
common.h
crc.h
crc.c
deal.h
deal.c
dhrand.h
dhrand.c
getnet.h
getnet.c
netserv.c

5.1 Procedures

chash.h - includes, defines, structures and procedure declarations that are used in chash.c
chash.c - procedures for making a consistent hash.

```
/* get a number of points for a machine */
int dhGetMachinePoints(char *hostname, int points);
```

```
/* add a machine,point pair to the unit interval */
int dhAddMachineToInterval(char *hostname, double d );

/* find the nearest machine to any point on the unit interval */
struct machplace * dhFindNearestMachine(double point);

/* construct the unit interval and fill it with hosts */
int dhBuildUI();

/* construct the tree until we find ourself in it */
int dhBuildTree(char *url, int n);

/* take a hostname, and make an edge out of it.
edge * dhMakeEdge(char *host);

/* take a url and find the immediate parent */
edge * dhGetParent(char *url);

/* return the number of neighbors */
int dhNeighborsCount(edge *dhparent);
```

common.h - definitions and structures that used in deal.c, getnet.c and netserv.c
crc.h - includes, defines, structures and procedure declarations that are used in crc.c
crc.c - procedures for generating a crc32 hash.

```
/* compute the crc32 hash for a string */
long crc32(char *buf, int len);

/* initialization procedure that must be called before crc32 */
void init_crc32();
```

deal.h - includes, defines, structures and procedure declarations that are used in deal.c
deal.c - Handle the requests made in netserv.c or getnet.c.

```
/* add a host to the host table, check for duplicates */
int addhost(char address[MAXADDR], time_t ntime, int np);
```

```
/* delete a host from the host table if it is there */
int delhost(char address[MAXADDR]);

/* print a list that represents the current network state */
int list(int newsockfd);

/* print out all the changes since time stime */
int since(int newsockfd, time_t stime);

/* dump the current network state to the dump file */
int dump();

/* read a dump file and construct a host table from it */
int readdump();
```

dhrand.h - includes, defines, structures and procedure declarations that are used in dhrand.c
dhrand.c - non-system specific version of rand and srand, so that we get reliable results across platforms

```
/* generate a pseudo-random number */
int dhRand();

/* seed for generating a pseudo-random number */
void dhSrand(seed);
```

getnet.h - includes, defines, structures and procedure declarations that are used in getnet.c
getnet.c - Get what other machines are on the network, and keep this updated.

```
/* get the initial network */
int getnet();

/* get the changes since ltime */
int getupdate(time_t ltime);

/* parse the log file and do the changes indicated */
int readupdate();
```

```
/* add this host to the network */
int addself();

/* delete this host from the network */
int delself();

/* open a socket to the server */
int sopen();
```

netserv.c - Separate program that will maintain and serve the list of who is on the network.

```
/* main loop. start up, wait for connections and deal with them */
int main(int argc, char *argv[]);

/* print a usage message */
void nsusage(char *name);

/* check what we received and call the appropriate function */
int deal(char caddr_new[MAXADDR], int newsockfd, time_t ntime,
char buf[MAXLINE], FILE *logPtr);

/* dump and exit on a signal */
int myexit();
```

A New code

A.1 chash.h

```
#ifndef CHASH_H
#define CHASH_H

/* the degree of the tree */
#define DEGREE_OF_TREE 2;

/* structure for a point, machine pair on unit interval */
struct machplace {
  double position;
  char *name;
  struct machplace *nextmach;
};
```

```
int dhGetMachinePoints _PARAMS((char *hostname, int points));
int dhAddMachineToInterval _PARAMS((char *hostname, double d ));
char * dhFindNearestMachine _PARAMS((double point));
int dhBuildUI _PARAMS(());
int dhBuildTree _PARAMS((char *url, int n));
edge * dhMakeEdge _PARAMS((char *host));
edge * dhGetParent _PARAMS((char *url));
int dhNeighborsCount _PARAMS((edge *dhparent));

#endif /* CHASH_H */
```

A.2 chash.c

```
#include "squid.h"

extern int num_hosts;
extern const char *myhostname;
extern struct hostinfo *tablePtr;
extern struct machplace *uiPtr;

/* get a number of points for a machine */
int
dhGetMachinePoints(char *hostname, int points) {
  int f;
  int len = (int) strlen(hostname);
  /* get the crc32 hash for that hostname */
  long init = crc32(hostname,len);
  /* and use that to seed the pseudo-random numbers */
  dhSrand(init);

  /* get one pseudo-random number between 0 and 1 for each point */
  for(f=0;f<points;f++) {
    unsigned int i = dhRand();
    double d = ((double)i) / ((double)DH_RAND_MAX);
    /* add the machine,point pair to the unit interval */
    dhAddMachineToInterval(hostname,d);
  }
  return 0;
}

/* add a machine,point pair to the unit interval */
int
dhAddMachineToInterval(char *hostname, double d ) {
  struct machplace *newmach, *lastPtr, *currPtr =uiPtr;

  /* go through the unit interval */
```

```
  while (currPtr != NULL) {

    /* save the current place for the next round */
    lastPtr = currPtr;

    /* deal with duplicates - they will happen eventually */
    if (currPtr->position == d) {
      int i;
      d = d + 0.000001;
      i = dhAddMachineToInterval(hostname,d);
      return i;
    }
    /* find the first higher position in the unit interval */
    else if (currPtr->position > d) {

      /* malloc space for a new entry in the table */
      newmach = (struct machplace *) malloc(sizeof(struct machplace));
      if (newmach == NULL){
            perror("malloc");
            return(-1);
      }
      /* malloc the space for the machine name */
      newmach->name = malloc(strlen(hostname) + 1);
      if (newmach->name == NULL) {
            perror("malloc");
            return(-1);
      }
      /* put the new address and time in the malloc'd space, and point the
        * entry's nexthost pointer to the next entry */
      strcpy(newmach->name,hostname);
      newmach->position = d;
      newmach->nextmach =  currPtr;

      lastPtr->nextmach = newmach;

      return(0);
    }
    /* increment to the next entry */
    currPtr = currPtr->nextmach;
  }
  /* this should never be reached. */
  return -1;
}

/* find the nearest machine to any point on the unit interval */
char *
dhFindNearestMachine(double point) {
```

```
  struct machplace *lastPtr, *currPtr =uiPtr;

  /* go through the unit interval */
  while (currPtr != NULL) {

    /* save the current place for the next round */
    lastPtr = currPtr;
    /* find the first higher point */
    if (currPtr->position >= point) {
      /* compute the differences between the two nearest points */
      double upper, lower;
      upper = currPtr->position - point;
      lower = lastPtr->position - point;
      /* return the closer point */
      if (upper > lower) {
            return lastPtr->name;
      }
      else {
            return currPtr->name;
      }
    }
  }
  /* if no point is higher, return the last point */
  return lastPtr->name;
}

/* construct the unit interval and fill it with hosts */
int
dhBuildUI() {
  struct hostinfo *thisPtr = tablePtr;
  int n=0;

  /* go through the host table and add the points for each machine */
  while (thisPtr != NULL) {
    dhGetMachinePoints(thisPtr->addr,3);
    n++;
    thisPtr = thisPtr->nexthost;
  }
  return n;
}

/* construct the tree until we find outself in it */
int
dhBuildTree(char *url, int n) {
  int i;
  /* loop through the number of hosts until we find the first occurance
   * of ourself */
```

```
  for (i=1;i>=n;i++) {
    char u[MAX_URL+ sizeof(long)];
    double d;
    char *name;
    /* calculate the point for that url,number pair */
    sprintf(u, "%d%s", i, url);
    d = crc32(u,sizeof(u)) % ((u_int)DH_RAND_MAX + 1);
    /* find which machine is closest to this point */
    name = dhFindNearestMachine(d);
    /* when we find ourself, return our position in the tree */
    if (name == myhostname) {
      return n;
    }
  }
  /* if we never find ourself */
  return 0;
}

/* take a hostname, and make an edge out of it. */
edge *
dhMakeEdge(char *host) {
  u_short http_port = CACHE_HTTP_PORT;
  u_short icp_port = CACHE_ICP_PORT;
  int options = 0;
  int weight = 1;
  int mcast_ttl = 0;

  edge *e = NULL;
  e = xcalloc(1, sizeof(edge));
  e->http_port = http_port;
  e->icp_port = icp_port;
  e->mcast_ttl = mcast_ttl;
  e->options = options;
  e->weight = weight;
  e->host = xstrdup(host);
  e->pinglist = NULL;
  e->typelist = NULL;
  e->acls = NULL;
  e->icp_version = ICP_VERSION_CURRENT;
  e->type = parseNeighborType("parent");

  return e;
}

/* take a url and find the immediate parent */
edge *
dhGetParent(char *url) {
```

```
  int myplace, parentplace;
  double d;
  char *name, u[MAX_URL+ sizeof(long)];
  edge *edge_answer;
  int deg = DEGREE_OF_TREE;


  myplace = dhBuildTree(url,num_hosts);

  if (myplace > 1) {
    parentplace = myplace / deg;

    sprintf(u, "%d%s", parentplace, url);
    d = crc32(u,sizeof(u)) % ((u_int)DH_RAND_MAX + 1);

    name = dhFindNearestMachine(d);
    edge_answer = dhMakeEdge(name);
    return edge_answer;
  }
  else {
    /* we are either at the top of the tree, or not in the tree at all */
    return NULL;
  }
}

/* return the number of neighbors */
int
dhNeighborsCount(edge *dhparent) {
  /* only one neighbor, else we are at the top of the tree, then none */
  if (dhparent == NULL) {
    return 0;
  }
  else {
    return 1;
  }
}
```

A.3 common.h

```
/* definitions that are used in multiple source files */

#ifndef _COMMON_H
#define _COMMON_H


#define MAXLINE 1024
#define MAXADDR 512
```

```
#define SERVER "localhost"
#define SERV_PORT 8999
#define GLOG_FILE "getnet.log"
#define DUMP_FILE "dump"
#define NLOG_FILE "netserv.log"
#define TIMEOUT 5
#define U_TIMEOUT 0
#define UPDATE_TIME 600

struct hostinfo {
  char addr[MAXADDR];
  time_t time;
  int num_points;
  struct hostinfo *nexthost;
};

struct list {
  int num;
  time_t ctime;
  struct list *next;
};

#endif /* COMMON_H */
```

A.4 crc.h

```
#ifndef CRC_H
#define CRC_H

void init_crc32 _PARAMS(());
long crc32 _PARAMS((char *buf, int len));

#endif /* CRC_H */
```

A.5 crc.c

```
#include "squid.h"

/* As its name implies (Cyclic Redundancy Check) a crc adds
redundancywhereas the topic of this group is to remove it. But since
thisquestion comes up often, here is some code (by Rob Warnock
<rpw3@sgi.com>).The following C code does CRC−32 in
BigEndian/BigEndian byte/bit order.That is, the data is sent most
significant byte first, and each of the bitswithin a byte is sent most
significant bit first, as in FDDI. You will needto twiddle with it to
do Ethernet CRC, i.e., BigEndian/LittleEndian byte/bitorder. [Left as
an exercise for the reader.]The CRCs this code generates agree with
```

```
the vendor-supplied Verilog modelsof several of the popular FDDI "MAC"
chips. */

long crc32_table[256];

long crc32(char *buf, int len) {
  char *p;
  long crc;
  if (!crc32_table[1]) /* if not already done, */
    init_crc32(); /* build table */

  crc = 0xffffffff; /* preload shift register, per CRC-32 spec */

  for (p = buf; len > 0; ++p, --len)
    crc = (crc << 8) ^ crc32_table[(crc >> 24) ^ *p];
  return ~crc; /* transmit complement, per CRC-32 spec */}

/* Build auxiliary table for parallel byte-at-a-time CRC-32. */

#define CRC32_POLY 0x04c11db7   /* AUTODIN II, Ethernet, & FDDI */

void init_crc32() {
  int i, j;
  long c;
  for (i = 0; i < 256; ++i) {
    for (c = i << 24, j = 8; j > 0; --j)
      c = c & 0x80000000 ? (c << 1) ^ CRC32_POLY : (c << 1);
    crc32_table[i] = c;
  }
}
```

A.6 deal.h

```
#ifndef DEAL_H
#define DEAL_H

int addhost(char address[1024], time_t ntime, int np);
int delhost(char address[1024]);
int list(int newsockfd);
int since(int newsockfd, time_t stime);
int dump();
int readdump();

#endif /* DEAL_H */
```

A.7 deal.c

```
#include "squid.h"

/* global variable that points to the main host table */
struct hostinfo *tablePtr;

/* add a host to the host table, check for duplicates */
int addhost(char address[MAXADDR], time_t ntime, int np){

  /* go through the table and check that the address is not already there */
  struct hostinfo *machinfo, *currPtr = tablePtr;
  while (currPtr != NULL) {
    if ((strcmp(currPtr->addr,address)) == 0){
      debug(1,0,"Duplicate address %s.\n",address);
      return(1);
    }
    else {
      currPtr = currPtr->nexthost;
    }
  }

  /* malloc space for a new entry in the table */
  machinfo = (struct hostinfo *) malloc(sizeof(struct hostinfo));
  if (machinfo == NULL){
    perror("malloc");
    return(-1);
  }
  else {
    /* put the new address and time in the malloc'd space, and point the
     * entry's nexthost pointer to the beginning of the old table */
    strcpy(machinfo->addr,address);
    machinfo->time = ntime;
    machinfo->num_points = np;
    machinfo->nexthost = tablePtr;
  }

  /* change the table to point to the new entry */
  tablePtr = machinfo;
  return(0);
}

/* delete a host from the host table if it is there */
int delhost(char address[MAXADDR]) {
  int found = 1;
  struct hostinfo *freePtr, **lastPtr = &tablePtr;

  /* go through the table, and if you find a match, point the previous entry's
   * nexthost pointer to the following entry, then free the memory that
```

```
  * contained the current entry */
 while (*lastPtr != NULL) {
  if ((strcmp((*lastPtr)->addr,address)) == 0) {
   freePtr = *lastPtr;
   *lastPtr = (*lastPtr)->nexthost;
   free(freePtr);

   /* set found to zero and stop looking through the table */
   found = 0;
   break;
  }
  lastPtr = &(*lastPtr)->nexthost;
 }

 /* return zero if the host is found, and one if it is not */
 return(found);
}

/* print a list that represents the current network state */
int list(int newsockfd) {
 int reply_len;
 char reply[MAXLINE] = "\0";
 struct hostinfo *currPtr = tablePtr;

 /* print header line */
 sprintf(reply, "200 Listing follows:\n");
 reply_len = strlen(reply);
 write(newsockfd,reply,reply_len);

 /* go through the host table and print out a line for every entry */
 while (currPtr != NULL) {
  sprintf(reply, "%s %d %d\n", currPtr->addr,(int) currPtr->time, currPtr->num_points);
  reply_len = strlen(reply);
  write(newsockfd,reply,reply_len);
  currPtr = currPtr->nexthost;
 }
 return(0);
}

/* print out all the changes since time stime */
int since(int newsockfd, time_t stime) {
 int reply_len;
 FILE *logPtr;
 char reply[MAXLINE] = "\0";
 char x[1], f[MAXADDR], s[MAXLINE];
 int *ftimePtr = malloc(sizeof(int));
```

```
  /* print the header line */
  sprintf(reply,"200 Changes since %i:\n",(int) stime);
  reply_len = strlen(reply);
  write(newsockfd,reply,reply_len);

  /* open the log file for reading */
  if ((logPtr = fopen(NLOG_FILE,"r")) == NULL) {
    perror("Error opening NLOG_FILE");
    exit(-1);
  }

  /* go through the log file and parse it */
  while (fgets(s,sizeof(s),logPtr) != NULL) {
    sscanf(s,"%s %s %d",x,f,ftimePtr);

    /* if the time the change was made is since stime, print out that line */
    if (stime <= *ftimePtr) {
      sprintf(reply, "%s",s);
      reply_len = strlen(reply);
      write(newsockfd,reply,reply_len);
    }
  }
  fclose(logPtr);
  return(0);
}

/* dump the current network state to the dump file */
int dump() {
  FILE *dumpPtr;
  int dumpfd;

  /* open the dump file to write */
  if ((dumpPtr = fopen(DUMP_FILE,"w")) == NULL) {
    perror("Couldn't open dump file");
    return (1);
  }
  dumpfd = fileno(dumpPtr);

  debug(1,0,"Dumping list.\n");

  /* do a list out to the dump file */
  if ((list(dumpfd)) != 0) {
    debug(1,0,"Error in dumping list.\n");
    return(1);
  }

  /* close the dump file */
```

```
  if((fclose(dumpPtr)) != 0) {
    perror("Couldn't close dump file");
    return(1);
  }
  return(0);
}

/* read a dump file and construct a host table from it */
int readdump() {
  FILE *dumpPtr;
  char line[MAXLINE];
  char addr[MAXADDR];
  int *np=malloc(sizeof(int)), *ptime=malloc(sizeof(int));

  /* open the dump file to read */
  if ((dumpPtr = fopen(DUMP_FILE,"r")) == NULL) {
    perror("Couldn't open dump file, continuing without it.");
    return(1);
  }

  /* get and throw away the header line */
  fgets(line,MAXLINE,dumpPtr);

  /* go through the rest of the file and add any hosts found */
  while((fgets(line,MAXLINE,dumpPtr)) != NULL) {
    sscanf(line,"%s %d %d",addr,ptime,np);
    addhost(addr,(time_t)*ptime,*np);
  }
  fclose(dumpPtr);
  return(0);
}
```

A.8 dhrand.h

```
#ifndef DHRAND_H
#define DHRAND_H

#define DH_RAND_MAX 0x7fffffff

int dhRand _PARAMS(());
void dhSrand _PARAMS((u_int seed));

#endif /* DHRAND_H */
```

A.9 dhrand.c

```
#include "squid.h"
```

```
static u_long next = 1;

/* generate a psuedo-random number */
int
dhRand()
{
        return ((next = next * 1103515245 + 12345) % ((u_int)DH_RAND_MAX + 1));
}

/* seed for generating a psuedo-random number */
void
dhSrand(u_int seed)
{
        next = seed;
}
```

A.10 getnet.h

```
#ifndef GETNET_H
#define GETNET_H


int sopen _PARAMS(());
int getnet _PARAMS(());
int getupdate _PARAMS((time_t ltime));
int readupdate _PARAMS(());
int addself _PARAMS(());
int delself _PARAMS(());
int myexit _PARAMS(());

#endif /* GETNET_H */
```

A.11 getnet.c

```
#include "squid.h"

/* default number of points to assign a host */
int opt_points = 1;

/* set the default servername from the .h file */
char opt_server[MAXADDR] = SERVER;

/* get the initial network */
int getnet() {
  int n, sockfd;
```

```
  char buf[MAXLINE];
  FILE *dumpPtr;
  int dumpfd;

  /* open the dump file to write */
  if ((dumpPtr = fopen(DUMP_FILE,"w")) == NULL) {
    perror("Couldn't open dump file");
    return (1);
  }
  dumpfd = fileno(dumpPtr);

  /* open a socket to the server */
  if ((sockfd = sopen()) == 0) {
    debug(1,0,"Couldn't open socket to server.\n");
    return(1);
  }

  debug(1,0,"Getting initial listing.\n");

  /* send the request */
  write(sockfd,"list\n",5);

  /* read the response and write it out the dump file */
  n = MAXLINE;
  while(n != 0) {
    n = read(sockfd,buf,MAXLINE);
    n = write(dumpfd,buf,n);
  }

  /* read the dump file that we have just written */
  readdump();
  close(sockfd);
  fclose(dumpPtr);
  return(0);
}

/* get the changes since ltime */
int getupdate(time_t ltime) {
  int sockfd, logfd, n;
  FILE *logPtr;
  char buf[MAXLINE];

  /* open the log file for writing */
  if ((logPtr = fopen(GLOG_FILE,"w")) == NULL) {
    perror("Can't open GLOG_FILE");
    return(1);
  }
```

```
  logfd = fileno(logPtr);

  /* open a socket to the server */
  if ((sockfd = sopen()) == 0) {
    debug(1,0,"Couldn't open socket to server.\n");
    return(1);
  }
  debug(1,0,"Getting update.\n");

  /* send the request */
  sprintf(buf,"since %d\n",(int) ltime);
  write(sockfd,buf,MAXLINE);

  /* read the response and write it out the log file */
  n = MAXLINE;
  while(n != 0) {
    n = read(sockfd,buf,MAXLINE);
    n = write(logfd,buf,n);
  }

  /* close the log file */
  fclose(logPtr);

  /* read the log file that we have just written */
  readupdate();
  close(sockfd);
  return(0);
}

/* parse the log file and do the changes indicated */
int readupdate() {
  int changed = 1;
  int *ptime = malloc(sizeof(int)), *np = malloc(sizeof(int));
  char line[MAXLINE], x[1], addr[MAXADDR];
  FILE *logPtr;

  /* open the log file for reading */
  if ((logPtr = fopen(GLOG_FILE,"r")) == NULL) {
    perror("Can't open GLOG_FILE");
    return(1);
  }

  /* get and throw away the header line */
  fgets(line,MAXLINE,logPtr);

  /* go through the rest of the log file and parse it */
  while((fgets(line,MAXLINE,logPtr)) != NULL) {
```

```
    sscanf(line,"%s %s %d %d",x,addr,ptime,np);

    /* if there were any changes, set this variable */
    changed = 0;

    /* if the line begins with a plus sign, add the host */
    if ((strcmp(x,"+")) == 0) {
      addhost(addr,(time_t)*ptime,*np);
      debug(1,0,"Adding %s %d %d.\n",addr,*ptime, *np);
    }

    /* if the line begins with a minus sign, delete the host */
    else if ((strcmp(x,"-")) == 0) {
      delhost(addr);
      debug(1,0,"Deleting %s.\n",addr);
    }

    /* otherwise, we have a badly formatted line */
    else {
      debug(1,0,"Error in parsing results from since.\n");
    }
  }

  /* if anything changed, do a dump */
  if (changed == 0) {
    debug(1,0,"Changes made, doing a dump.\n");
    dump();
  }

  /* close the log file */
  fclose(logPtr);
  return(0);
}

/* add this host to the network */
int addself() {
  int sockfd, message_len;
  char message[MAXLINE], buf[MAXLINE];

  /* open a socket to the server */
  if ((sockfd = sopen()) == 0) {
    debug(1,0,"Couldn't open socket to server.\n");
    return(1);
  }
  debug(1,0,"Joining network.\n");

  /* message is join and the number of points to grab */
```

```
  sprintf(message,"join %d\n",opt_points);
  message_len = strlen(message);

  /* send the request */
  write(sockfd,message,message_len);

  /* read the response */
  read(sockfd,buf,MAXLINE);
  debug(1,0,"%s",buf);

  close(sockfd);
  return(0);
}

/* delete this host from the network */
int delself() {
  int sockfd;
  char buf[MAXLINE];

  /* open a socket to the server */
  if ((sockfd = sopen()) == 0) {
    debug(1,0,"Couldn't open socket to server.\n");
    return(1);
  }
  debug(1,0,"Leaving network.\n");

  /* send the request */
  write(sockfd,"dele\n",5);

  /* read the response */
  read(sockfd,buf,MAXLINE);
  debug(1,0,"%s",buf);
  close(sockfd);
  return(0);
}

/* open a socket to the server */
int sopen() {
  int sockfd;
  struct sockaddr_in serv_addr;
  struct hostent *hostPtr;
  int one = 1;

  /* translate the server's name to an IP address */
  if ((hostPtr = gethostbyname(opt_server)) == NULL) {
    perror("gethostbyname");
    return(0);
```

```
  }

  /* open the socket */
  if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
    perror("socket");
    return(0);
  }

  /* clear out and set the struct for the server address */
  memset(&serv_addr, 0, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  memcpy(&serv_addr.sin_addr, hostPtr->h_addr,sizeof(struct in_addr));
  serv_addr.sin_port = htons(SERV_PORT);

  /* allow the socket to be reused immediately */
  setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (char *)&one, sizeof(one));

  /* connect to the server */
  if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0) {
    perror("connect");
    return(0);
  }
  return(sockfd);
}
```

A.12 netserv.c

```
#include "squid.h"

/* global variable that points to the main host table */
struct hostinfo *tablePtr = NULL;

/* default number of points to assign a host */
int opt_points = 1;

/* Junk so we can link with debug.o */
int opt_syslog_enable = 0;
volatile int unbuffered_logs = 1;
const char *const appname = "netserv";
struct timeval current_time;
time_t squid_curtime;
struct SquidConfig Config;

void nsusage _PARAMS((char *name));
int deal _PARAMS((char caddr_new[MAXADDR], int newsockfd, time_t ntime, char buf[MAXLINE], FILE *logPtr));
int myexit _PARAMS(());
```

```
/* main loop. start up, wait for connections and deal with them */
int main(int argc, char *argv[])
{
  int sockfd, newsockfd, cli_len, count;
  struct sockaddr_in serv_addr, cli_addr;
  int one = 1;
  FILE *logPtr;
  struct list *fdlist = NULL;
  struct timeval timeout;
  extern char *optarg;
  int c;

  while ((c = getopt(argc, argv, "hp:?")) != -1) {
    switch (c) {
    case 'h':
      nsusage(argv[0]);
      break;
    case 'p':
      opt_points = atoi(optarg);
      break;
    case '?':
    default:
      nsusage(argv[0]);
      break;
    }
  }
  /* debugging message */
  debug(1,0,"Starting up...\n");

  /* open the log file for appending */
  if ((logPtr = fopen(NLOG_FILE,"a")) == NULL) {
    debug(1,0,"Can't open log file %s.\n",NLOG_FILE);
    exit(-1);
  }

  /* open a socket */
  if( (sockfd = socket(AF_INET, SOCK_STREAM,0)) < 0) {
    perror("socket");
    exit(-1);
  }

  /* clear out and set the struct for the server address */
  memset(&serv_addr, 0, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  serv_addr.sin_port = htons(SERV_PORT);
```

```
/* allow the socket to be reused immediately */
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (char *)&one, sizeof(one));

/* bind to the socket */
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
  perror("bind");
  exit(-1);
}

/* list on the socket */
listen(sockfd,5);

/* read in the saved network status from the dump file */
readdump();

/* catch SIGINT and SIGHUP and exit if you get them */
signal(SIGINT,(void *) myexit);
signal(SIGHUP,(void *) myexit);

/* select loop */
for(;;){
  fd_set fdvar;
  struct list *listPtr, **lastPtr, *currPtr = fdlist;

  FD_ZERO(&fdvar);
  FD_SET(sockfd,&fdvar);

  while (currPtr != NULL) {
    FD_SET(currPtr->num,&fdvar);
    currPtr = currPtr->next;
  }

  /* set the timeout to wait for the client to send something */
  timeout.tv_sec = TIMEOUT;
  timeout.tv_usec = U_TIMEOUT;

  /* check what file descriptors need managing */
  count = select(NOFILE,(fd_set *)&fdvar,(fd_set *)0,(fd_set *)0,&timeout);
  if (count < 0) {
    perror("select");
    exit(-1);
  }
  if (count == 0){
    time_t currtime =  time((time_t *)0);
    lastPtr = &fdlist;
    while (*lastPtr != NULL) {
          if ((currtime - TIMEOUT) > (*lastPtr)->ctime) {
```

```
        debug(1,0,"Timeout on sockfd %d.\n",(*lastPtr)->num);
        close((*lastPtr)->num);
        currPtr= *lastPtr;
        *lastPtr = (*lastPtr)->next;
        free(currPtr);
    }
    else {
        lastPtr = &(*lastPtr)->next;
    }
  }
}
else {
  char caddr_new[MAXADDR];
  struct hostent *hp;
  if (FD_ISSET(sockfd,&fdvar)) {
      cli_len = sizeof(cli_addr);
      if ((newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
                                   &cli_len)) < 0) {
        perror("accept");
        exit(-1);
      }
      /* convert the IP address of the host that connected to its cname. */
      hp = gethostbyaddr((const char *) &cli_addr.sin_addr, sizeof(cli_addr.sin_addr), AF_INET);
      strcpy(caddr_new,hp->h_name);

      debug(1,0,"%s connected.\n",caddr_new);

      listPtr = (struct list *) malloc(sizeof(struct list));
      listPtr->num = newsockfd;
      listPtr->ctime = time((time_t *)0);
      listPtr->next = fdlist;
      fdlist = listPtr;
  }

  lastPtr = &fdlist;
  while (*lastPtr != NULL) {
      if (FD_ISSET((*lastPtr)->num,&fdvar)) {
        char buf[MAXLINE] = "\0";
        read((*lastPtr)->num, buf, MAXLINE);
        deal(caddr_new, (*lastPtr)->num, (*lastPtr)->ctime,buf,logPtr);
        close((*lastPtr)->num);
        currPtr= *lastPtr;
        *lastPtr = (*lastPtr)->next;
        free(currPtr);
      }
      else {
        lastPtr = &(*lastPtr)->next;
```

```
        }
      }
    }
  }
  close(sockfd);
  fclose(logPtr);
}

/* print a usage message */
void nsusage(char *name) {
  fprintf(stderr, "Usage:  %s [-h] [-p points]\n"
        "       -p points Set the default number of points.\n"
        "       -h        Get this help message.\n",name);
  myexit();
}

/* check what we received and call the appropriate function */
int deal(char caddr_new[MAXADDR], int newsockfd, time_t ntime, char buf[MAXLINE], FILE *logPtr) {
  int reply_len, retval;
  char reply[MAXLINE] = "\0";
  ntime = time((time_t *)0);

  /* check for a join request */
  if ((strncmp(buf,"join",4)) == 0) {

    /* get the number of points if given */
    int num;
    char s[5];
    if (sscanf(buf, "%s %d", s, &num) != 2) {
      num = opt_points;
    }
    /* try to add the host to the network */
    retval = addhost(caddr_new,ntime,num);

    /* if the host was added... */
    if ( retval == 0) {

      /* Write this to the log file and flush it */
      fprintf(logPtr,"+ %s %d %d\n",caddr_new,(int) ntime, num);
      fflush(logPtr);

      /* reply to the client */
      sprintf(reply, "200 You have been added to the network.\n");
      reply_len = strlen(reply);
      write(newsockfd,reply,reply_len);
      debug(1,0,"%s joined at %d with %d points.\n",caddr_new,(int) ntime,num);
    }
```

```
  /* if the host is already in the network, tell the client that */
  else if (retval == 1){
    sprintf(reply, "401 Duplicate address, not added.\n");
    reply_len = strlen(reply);
    write(newsockfd,reply,reply_len);
    debug(1,0,"Duplicate request from %s at %d.\n",caddr_new,(int) ntime);
  }
  /* for other errors */
  else {
    debug(1,0,"Couldn't add new host %s to network at %d.\n", caddr_new,(int) ntime);
  }
}

/* check for dele requests */
else if ((strncmp(buf,"dele",4)) == 0) {

  /* try to delete the host from the network */
  retval = delhost(caddr_new);

  /* if it was successfully deleted... */
  if ( retval == 0 ){

    /* write this to the log file and flush it */
    fprintf(logPtr,"- %s %d\n",caddr_new,(int) ntime);
    fflush(logPtr);

    /* reply to the client */
    sprintf(reply, "200 You have been deleted from the network.\n");
    reply_len = strlen(reply);
    write(newsockfd,reply,reply_len);
    debug(1,0,"%s deleted at %d.\n",caddr_new,(int) ntime);
  }

  /* if the address was not in the network, tell the client that */
 else if (retval == 1){
    sprintf(reply, "401 No such address, not deleted.\n");
    reply_len = strlen(reply);
    write(newsockfd,reply,reply_len);
    debug(1,0,"No such address to delete as %s at %d.\n",caddr_new,(int) ntime);
  }

  /* for other errors */
  else {
    debug(1,0,"Couldn't delete host %s from network at %d.\n",caddr_new,(int) ntime);
  }
}
```

```
/* check for list requests */
else if ((strncmp(buf,"list",4)) == 0){

  /* print the list out to this connection */
  if ((list(newsockfd)) == 0){
    debug(1,0,"%s requested a listing at %d.\n",caddr_new,(int) ntime);
  }
}

/* check for help requests */
else if ((strncmp(buf,"help",4)) == 0){

  /* reply to the client */
  sprintf(reply,"100 Possible commands are:\n\t join [<points>]\n\t dele\n\t list\n\t since <time>\n\t help\n");
  reply_len = strlen(reply);
  write(newsockfd,reply,reply_len);
  debug(1,0,"Help request from %s at %d.\n",caddr_new,(int) ntime);
}

/* check for since requests */
else if ((strncmp(buf,"since",5)) == 0) {
  /* get the time argument */
  int *stimePtr = malloc(sizeof(int));
  char s[6];
  if (sscanf(buf,"%s %d",s,stimePtr) == 2) {

    /* print the since information out to this connection */
    if ((since(newsockfd,*stimePtr)) == 0) {
          debug(1,0,"Since %d request from %s at %d.\n",*stimePtr,caddr_new,(int) ntime);
    }
  }

  /* for other errors */
  else {
    sprintf(reply, "400 Invalid since request.\n");
    reply_len = strlen(reply);
    write(newsockfd,reply,reply_len);
    debug(1,0,"Invalid since request from %s at %d.\n",caddr_new,(int) ntime);
  }
}

/* otherwise, we have an unknown request */
else {
  /* reply to the client */
  sprintf(reply, "400 Unknown request.\n");
  reply_len = strlen(reply);
```

```
    write(newsockfd,reply,reply_len);
    debug(1,0,"Unknown request from %s at %d.\n",caddr_new,(int) ntime);
  }
  return(0);
}


/* dump and exit on a signal */
int myexit() {
  debug(1,0,"exiting.\n");
  if ((dump()) != 0) {
    debug(1,0,"Error in dumping network status, data may be lost.\n");
  }
  exit(0);
}
```

B Modified Squid Code

B.1 squid.h

```
/*
 * $Id: squid.h,v 1.4 1997/01/08 04:56:58 marthag Exp $
 *
 * AUTHOR: Duane Wessels
 *
 * SQUID Internet Object Cache  http://squid.nlanr.net/Squid/
 * --------------------------------------------------------------------------
 *
 *  Squid is the result of efforts by numerous individuals from the
 *  Internet community.  Development is led by Duane Wessels of the
 *  National Laboratory for Applied Network Research and funded by
 *  the National Science Foundation.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
```

```c
 *  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */

#ifndef SQUID_H
#define SQUID_H

#include "config.h"

/*
 * On some systems, FD_SETSIZE is set to something lower than the
 * actual number of files which can be opened.  IRIX is one case,
 * NetBSD is another.  So here we increase FD_SETSIZE to our
 * configure-discovered maximum *before* any system includes.
 */

/*
 * Linux (2.x only?) always defines FD_SETSIZE in <linux/time.h> so
 * we get in trouble if we try to increase it.  barf.
 */

#ifndef _SQUID_LINUX_
#if SQUID_MAXFD > FD_SETSIZE
#define FD_SETSIZE SQUID_MAXFD
#endif
#endif /* _SQUID_LINUX_ */

#if HAVE_UNISTD_H
#include <unistd.h>
#endif
#if HAVE_STDLIB_H
#include <stdlib.h>
#endif
#if HAVE_STDIO_H
#include <stdio.h>
#endif
#if HAVE_SYS_TYPES_H
#include <sys/types.h>
#endif
#if HAVE_CTYPE_H
#include <ctype.h>
#endif
#if HAVE_ERRNO_H
#include <errno.h>
#endif
#if HAVE_FCNTL_H
#include <fcntl.h>
```

```
#endif
#if HAVE_GRP_H
#include <grp.h>
#endif
#if HAVE_MALLOC_H && !defined(_SQUID_FREEBSD_) && !defined(_SQUID_NEXT_)
#include <malloc.h>
#endif
#if HAVE_MEMORY_H
#include <memory.h>
#endif
#if HAVE_NETDB_H && !defined(_SQUID_NETDB_H_)  /* protect NEXTSTEP */
#define _SQUID_NETDB_H_
#ifdef _SQUID_NEXT_
#include <netinet/in_systm.h>
#endif
#include <netdb.h>
#endif
#if HAVE_PWD_H
#include <pwd.h>
#endif
#if HAVE_SIGNAL_H
#include <signal.h>
#endif
#if HAVE_TIME_H
#include <time.h>
#endif
#if HAVE_SYS_PARAM_H
#include <sys/param.h>
#endif
#if HAVE_SYS_TIME_H
#include <sys/time.h>
#endif
#if HAVE_SYS_RESOURCE_H
#include <sys/resource.h>          /* needs sys/time.h above it */
#endif
#if HAVE_SYS_SOCKET_H
#include <sys/socket.h>
#endif
#if HAVE_NETINET_IN_H
#include <netinet/in.h>
#endif
#if HAVE_ARPA_INET_H
#include <arpa/inet.h>
#endif
#if HAVE_SYS_STAT_H
#include <sys/stat.h>
#endif
```

```
#if HAVE_SYS_UN_H
#include <sys/un.h>
#endif
#if HAVE_SYS_WAIT_H
#include <sys/wait.h>
#endif
#if HAVE_LIBC_H
#include <libc.h>
#endif
#ifdef HAVE_SYS_SYSCALL_H
#include <sys/syscall.h>
#endif
#ifdef HAVE_STRING_H
#include <string.h>
#endif
#ifdef HAVE_STRINGS_H
#include <strings.h>
#endif
#if HAVE_BSTRING_H
#include <bstring.h>
#endif
#ifdef HAVE_CRYPT_H
#include <crypt.h>
#endif
#if HAVE_SYS_SELECT_H
#include <sys/select.h>
#endif
#if HAVE_GETOPT_H
#include <getopt.h>
#endif

#if defined(USE_POLL) && HAVE_POLL
#if HAVE_POLL_H
#include <poll.h>
#endif /* HAVE_POLL_H */
#else
#undef USE_POLL
#endif

#ifdef __STDC__
#include <stdarg.h>
#else
#include <varargs.h>
#endif

/* Make sure syslog goes after stdarg/varargs */
#ifdef HAVE_SYSLOG_H
```

```
#include <syslog.h>
#endif

#if HAVE_SHADOW_H
#include <shadow.h>
#endif

#if !defined(MAXHOSTNAMELEN) || (MAXHOSTNAMELEN < 128)
#define SQUIDHOSTNAMELEN 128
#else
#define SQUIDHOSTNAMELEN MAXHOSTNAMELEN
#endif

#define SQUID_MAXPATHLEN 256
#ifndef MAXPATHLEN
#define MAXPATHLEN SQUID_MAXPATHLEN
#endif

#if !defined(HAVE_GETRUSAGE) && defined(_SQUID_HPUX_)
#define HAVE_GETRUSAGE 1
#define getrusage(a, b)  syscall(SYS_GETRUSAGE, a, b)
#endif

#if !defined(HAVE_GETPAGESIZE) && defined(_SQUID_HPUX_)
#define HAVE_GETPAGESIZE
#define getpagesize( )   sysconf(_SC_PAGE_SIZE)
#endif

#ifndef BUFSIZ
#define BUFSIZ  4096                /* make reasonable guess */
#endif

#ifndef SA_RESTART
#define SA_RESTART 0
#endif
#ifndef SA_NODEFER
#define SA_NODEFER 0
#endif
#ifndef SA_RESETHAND
#define SA_RESETHAND 0
#endif
#if SA_RESETHAND == 0 && defined(SA_ONESHOT)
#undef SA_RESETHAND
#define SA_RESETHAND SA_ONESHOT
#endif

typedef struct sentry StoreEntry;
```

```
typedef struct mem_hdr *mem_ptr;
typedef struct _edge edge;
typedef struct icp_common_s icp_common_t;
typedef struct _cacheinfo cacheinfo;
typedef struct _aclCheck_t aclCheck_t;
typedef struct _request request_t;
typedef struct _MemObject MemObject;
typedef struct _cachemgr_passwd cachemgr_passwd;

/* 32 bit integer compatability hack */
#if SIZEOF_INT == 4
typedef int num32;
typedef unsigned int u_num32;
#elif SIZEOF_LONG == 4
typedef long num32;
typedef unsigned long u_num32;
#else
typedef long num32;                /* assume that long's are 32bit */
typedef unsigned long u_num32;
#endif
#define NUM32LEN sizeof(num32)  /* this should always be 4 */

#if PURIFY
#define LOCAL_ARRAY(type,name,size) \
        static type *local_##name=NULL; \
        type *name = local_##name ? local_##name : \
                ( local_##name = (type *)xcalloc(size, sizeof(type)) )
#else
#define LOCAL_ARRAY(type,name,size) static type name[size]
#endif

#include "ansiproto.h"

#if HAVE_REGEX_H
#include <regex.h>
#else /* HAVE_REGEX_H */
#include "GNUregex.h"
#endif /* HAVE_REGEX_H */

/*
 * So if FD_SETSIZE is less than SQUID_MAXFD we'd probably better
 * shrink SQUID_MAXFD so that select(2) doesn't puke even though
 * we might really be able to open more than FD_SETSIZE descriptors.
 * happy happy happy joy joy joy.
 */
#if FD_SETSIZE < SQUID_MAXFD
#undef SQUID_MAXFD
```

```
#define SQUID_MAXFD FD_SETSIZE
#endif

typedef void (*SIH) (int, void *);          /* swap in */
typedef int (*QS) (const void *, const void *);

#include "cache_cf.h"
#include "comm.h"
#include "debug.h"
#include "disk.h"
#include "fdstat.h"
#include "filemap.h"
#include "hash.h"
#include "proto.h"              /* must go before neighbors.h */
#include "neighbors.h"          /* must go before url.h */
/* DH Additions to original SQUID cache release */
#include "chash.h"
#include "crc.h"
#include "common.h"
#include "deal.h"
#include "dhrand.h"
#include "getnet.h"
/* end DH changes */
#include "url.h"
#include "icp.h"
#include "errorpage.h"          /* must go after icp.h */
#include "dns.h"
#include "ipcache.h"
#include "fqdncache.h"
#include "mime.h"
#include "stack.h"
#include "stat.h"
#include "stmem.h"
#include "store.h"
#include "tools.h"
#include "http.h"
#include "ftp.h"
#include "gopher.h"
#include "util.h"
#include "event.h"
#include "acl.h"
#include "async_io.h"
#include "redirect.h"
#include "client_side.h"
#include "useragent.h"
#include "icmp.h"
#include "net_db.h"
```

```
#include "client_db.h"
#include "objcache.h"
#include "refresh.h"

#if !HAVE_TEMPNAM
#include "tempnam.h"
#endif

extern void serverConnectionsClose _PARAMS((void));
extern void shut_down _PARAMS((int));


extern time_t squid_starttime;      /* main.c */
extern int do_reuse;                /* main.c */
extern int theHttpConnection;       /* main.c */
extern int theInIcpConnection;      /* main.c */
extern int theOutIcpConnection;     /* main.c */
extern int vizSock;
extern volatile int shutdown_pending;       /* main.c */
extern volatile int reread_pending;         /* main.c */
extern int opt_unlink_on_reload;            /* main.c */
extern int opt_reload_hit_only;     /* main.c */
extern int opt_dns_tests;           /* main.c */
extern int opt_foreground_rebuild;          /* main.c */
extern int opt_zap_disk_store;      /* main.c */
extern int opt_syslog_enable;       /* main.c */
extern int opt_catch_signals;       /* main.c */
extern int opt_no_ipcache;          /* main.c */
/* begin DH changes */
extern int opt_dynamic_hierarchy; /* main.c - DH option */
/* end DH changes */
extern int vhost_mode;              /* main.c */
extern const char *const version_string;            /* main.c */
extern const char *const appname;           /* main.c */
extern struct in_addr local_addr;           /* main.c */
extern struct in_addr theOutICPAddr;        /* main.c */
extern const char *const localhost;
extern unsigned int inaddr_none;
extern struct in_addr any_addr;     /* comm.c */
extern struct in_addr no_addr;      /* comm.c */
extern int opt_udp_hit_obj;         /* main.c */
extern int opt_mem_pools;           /* main.c */
extern int opt_forwarded_for;       /* main.c */
extern int opt_accel_uses_host;     /* main.c */
extern char ThisCache[];            /* main.c */

/* Prototypes and definitions which don't really deserve a seaprate
```

```
 * include file */

#define CONNECT_PORT        443

extern int objcacheStart _PARAMS((int, const char *, StoreEntry *));
extern void send_announce _PARAMS((void *unused));
extern int sslStart _PARAMS((int fd, const char *, request_t *, char *, int *sz));
extern const char *storeToString _PARAMS((const StoreEntry *));
extern int waisStart _PARAMS((int, const char *, method_t, char *, StoreEntry *));
extern void storeDirClean _PARAMS((void *unused));
extern int passStart _PARAMS((int fd,
        const char *url,
        request_t * request,
        char *buf,
        int buflen,
        int *size_ptr));
extern void identStart _PARAMS((int, icpStateData *,
        void        (*callback) _PARAMS((void *))));

extern const char *const dash_str;
extern const char *const null_string;

#define OR(A,B) (A ? A : B)

#endif /* SQUID_H */
```

B.2 main.c

```
/*
 * $Id: main.c,v 1.3 1997/01/08 04:56:46 marthag Exp $
 *
 * DEBUG: section 1     Startup and Main Loop
 * AUTHOR: Harvest Derived
 *
 * SQUID Internet Object Cache  http://squid.nlanr.net/Squid/
 * --------------------------------------------------------------
 *
 *  Squid is the result of efforts by numerous individuals from the
 *  Internet community.  Development is led by Duane Wessels of the
 *  National Laboratory for Applied Network Research and funded by
 *  the National Science Foundation.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
```

```
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */

/*
 * Copyright (c) 1994, 1995.  All rights reserved.
 *
 *   The Harvest software was developed by the Internet Research Task
 *   Force Research Group on Resource Discovery (IRTF-RD):
 *
 *         Mic Bowman of Transarc Corporation.
 *         Peter Danzig of the University of Southern California.
 *         Darren R. Hardy of the University of Colorado at Boulder.
 *         Udi Manber of the University of Arizona.
 *         Michael F. Schwartz of the University of Colorado at Boulder.
 *         Duane Wessels of the University of Colorado at Boulder.
 *
 *   This copyright notice applies to software in the Harvest
 *   "src/" directory only.  Users should consult the individual
 *   copyright notices in the "components/" subdirectories for
 *   copyright information about other software bundled with the
 *   Harvest source code distribution.
 *
 * TERMS OF USE
 *
 *   The Harvest software may be used and re-distributed without
 *   charge, provided that the software origin and research team are
 *   cited in any use of the system.  Most commonly this is
 *   accomplished by including a link to the Harvest Home Page
 *   (http://harvest.cs.colorado.edu/) from the query page of any
 *   Broker you deploy, as well as in the query result pages.  These
 *   links are generated automatically by the standard Broker
 *   software distribution.
 *
 *   The Harvest software is provided "as is", without express or
 *   implied warranty, and with no support nor obligation to assist
 *   in its use, correction, modification or enhancement.  We assume
 *   no liability with respect to the infringement of copyrights,
 *   trade secrets, or any patents, and are not responsible for
 *   consequential damages.  Proper use of the Harvest software is
```

```
 *   entirely the responsibility of the user.
 *
 * DERIVATIVE WORKS
 *
 *   Users may make derivative works from the Harvest software, subject
 *   to the following constraints:
 *
 *     - You must include the above copyright notice and these
 *       accompanying paragraphs in all forms of derivative works,
 *       and any documentation and other materials related to such
 *       distribution and use acknowledge that the software was
 *       developed at the above institutions.
 *
 *     - You must notify IRTF-RD regarding your distribution of
 *       the derivative work.
 *
 *     - You must clearly notify users that your are distributing
 *       a modified version and not the original Harvest software.
 *
 *     - Any derivative product is also subject to these copyright
 *       and use restrictions.
 *
 *   Note that the Harvest software is NOT in the public domain.  We
 *   retain copyright, as specified above.
 *
 * HISTORY OF FREE SOFTWARE STATUS
 *
 *   Originally we required sites to license the software in cases
 *   where they were going to build commercial products/services
 *   around Harvest.  In June 1995 we changed this policy.  We now
 *   allow people to use the core Harvest software (the code found in
 *   the Harvest "src/" directory) for free.  We made this change
 *   in the interest of encouraging the widest possible deployment of
 *   the technology.  The Harvest software is really a reference
 *   implementation of a set of protocols and formats, some of which
 *   we intend to standardize.  We encourage commercial
 *   re-implementations of code complying to this set of standards.
 */

#include "squid.h"

time_t squid_starttime = 0;
int theHttpConnection = -1;
int theInIcpConnection = -1;
int theOutIcpConnection = -1;
int vizSock = -1;
int do_reuse = 1;
```

```
int opt_unlink_on_reload = 0;
int opt_reload_hit_only = 0;          /* only UDP_HIT during store relaod */
int opt_catch_signals = 1;
int opt_dns_tests = 1;
int opt_foreground_rebuild = 0;
int opt_zap_disk_store = 0;
int opt_syslog_enable = 0;            /* disabled by default */
int opt_no_ipcache = 0;               /* use ipcache by default */
static int opt_send_signal = -1;          /* no signal to send */
int opt_udp_hit_obj = 0;              /* ask for HIT_OBJ's */
int opt_mem_pools = 1;
int opt_forwarded_for = 1;
int opt_accel_uses_host = 0;
/* begin DH changes */
int opt_dynamic_hierarchy = 0;   /* by default, don't use dynamic hierarchy */
int num_hosts = 0;
struct hostinfo *tablePtr  = NULL;
/* global variable that points to the list of machines in the unit interval */
struct machplace *uiPtr = NULL;
const char *myhostname;
/* end DH changes */
int vhost_mode = 0;
volatile int unbuffered_logs = 1;         /* debug and hierarchy unbuffered by default */
volatile int shutdown_pending = 0;        /* set by SIGTERM handler (shut_down()) */
volatile int reread_pending = 0;          /* set by SIGHUP handler */
const char *const version_string = SQUID_VERSION;
const char *const appname = "squid";
const char *const localhost = "127.0.0.1";
struct in_addr local_addr;
struct in_addr no_addr;
struct in_addr any_addr;
struct in_addr theOutICPAddr;
const char *const dash_str = "-";
const char *const null_string = "";
char ThisCache[SQUIDHOSTNAMELEN << 1];
unsigned int inaddr_none;

/* for error reporting from xmalloc and friends */
extern void (*failure_notify) _PARAMS((const char *));

static volatile int rotate_pending = 0;           /* set by SIGUSR1 handler */
static int httpPortNumOverride = 1;
static int icpPortNumOverride = 1;        /* Want to detect "-u 0" */
#if MALLOC_DBG
static int malloc_debug_level = 0;
#endif
```

```
static void rotate_logs _PARAMS((int));
static void reconfigure _PARAMS((int));
static void mainInitialize _PARAMS((void));
static void mainReinitialize _PARAMS((void));
static void usage _PARAMS((void));
static void mainParseOptions _PARAMS((int, char **));
static void sendSignal _PARAMS((void));
static void serverConnectionsOpen _PARAMS((void));

static void
usage(void)
{
    fprintf(stderr,
            /* begin DH changes */
            "Usage:  %s [-dhsvzCDFRUVY] [-f config-file] [-[au] port] [-k signal]\n"
        "       -a port   Specify ASCII port number (default:  %d).\n"
        "       -d        Use dynamic hierarchy.\n"
            /* end DH changes */
        "       -f file   Use given config-file instead of\n"
        "                 %s\n"
        "       -h        Print help message.\n"
        "       -i        Disable IP caching.\n"
        "       -k reconfigure|rotate|shutdown|interrupt|kill|debug|check\n"
        "                 Send signal to running copy and exit.\n"
        "       -s        Enable logging to syslog.\n"
        "       -u port   Specify ICP port number (default:  %d), disable with 0.\n"
        "       -v        Print version.\n"
        "       -z        Zap disk storage -- deletes all objects in disk cache.\n"
        "       -C        Do not catch fatal signals.\n"
        "       -D        Disable initial DNS tests.\n"
        "       -F        Foreground fast store rebuild.\n"
        "       -R        Do not set REUSEADDR on port.\n"
        "       -U        Unlink expired objects on reload.\n"
        "       -V        Virtual host httpd-accelerator.\n"
        "       -Y        Only return UDP_HIT or UDP_RELOADING during fast reload.\n",
        appname, CACHE_HTTP_PORT, DefaultConfigFile, CACHE_ICP_PORT);
    exit(1);
}

static void
mainParseOptions(int argc, char *argv[])
{
    extern char *optarg;
    int c;

    /* begin DH changes */
    while ((c = getopt(argc, argv, "CDFRUVYa:bdf:hik:m:su:vz?")) != -1) {
```

```
/* end DH changes */
    switch (c) {
    case 'C':
        opt_catch_signals = 0;
        break;
    case 'D':
        opt_dns_tests = 0;
        break;
    case 'F':
        opt_foreground_rebuild = 1;
        break;
    case 'R':
        do_reuse = 0;
        break;
    case 'U':
        opt_unlink_on_reload = 1;
        break;
    case 'V':
        vhost_mode = 1;
        break;
    case 'Y':
        opt_reload_hit_only = 1;
        break;
    case 'a':
        httpPortNumOverride = atoi(optarg);
        break;
    case 'b':
        unbuffered_logs = 0;
        break;
        /* begin DH changes */
    case 'd':
        opt_dynamic_hierarchy = 0;
        break;
    /* end DH changes */
    case 'f':
        xfree(ConfigFile);
        ConfigFile = xstrdup(optarg);
        break;
    case 'h':
        usage();
        break;
    case 'i':
        opt_no_ipcache = 1;
        break;
    case 'k':
        if ((int) strlen(optarg) < 1)
                usage();
```

```
            if (!strncmp(optarg, "reconfigure", strlen(optarg)))
                opt_send_signal = SIGHUP;
            else if (!strncmp(optarg, "rotate", strlen(optarg)))
                opt_send_signal = SIGUSR1;
            else if (!strncmp(optarg, "debug", strlen(optarg)))
                opt_send_signal = SIGUSR2;
            else if (!strncmp(optarg, "shutdown", strlen(optarg)))
                opt_send_signal = SIGTERM;
            else if (!strncmp(optarg, "interrupt", strlen(optarg)))
                opt_send_signal = SIGINT;
            else if (!strncmp(optarg, "kill", strlen(optarg)))
                opt_send_signal = SIGKILL;
            else if (!strncmp(optarg, "check", strlen(optarg)))
                opt_send_signal = 0;      /* SIGNULL */
            else
                usage();
            break;
        case 'm':
#if MALLOC_DBG
            malloc_debug_level = atoi(optarg);
            /* NOTREACHED */
            break;
#else
            fatal("Need to add -DMALLOC_DBG when compiling to use -m option");
            /* NOTREACHED */
#endif
        case 's':
            opt_syslog_enable = 1;
            break;
        case 'u':
            icpPortNumOverride = atoi(optarg);
            if (icpPortNumOverride < 0)
                icpPortNumOverride = 0;
            break;
        case 'v':
            printf("Squid Cache:  Version %s\n", version_string);
            exit(0);
            /* NOTREACHED */
        case 'z':
            opt_zap_disk_store = 1;
            break;
        case '?':
        default:
            usage();
            break;
        }
    }
```

```
}

static void
rotate_logs(int sig)
{
    debug(1, 1, "rotate_logs:  SIGUSR1 received.\n");
    rotate_pending = 1;
#if !HAVE_SIGACTION
    signal(sig, rotate_logs);
#endif
}

static void
reconfigure(int sig)
{
    debug(1, 1, "reconfigure:  SIGHUP received\n");
    debug(1, 1, "Waiting %d seconds for active connections to finish\n",
        Config.lifetimeShutdown);
    reread_pending = 1;
#if !HAVE_SIGACTION
    signal(sig, reconfigure);
#endif
}

void
shut_down(int sig)
{
    shutdown_pending = sig == SIGINT ? -1 : 1;
    debug(1, 1, "Preparing for shutdown after %d connections\n",
        ntcpconn + nudpconn);
    if (opt_dynamic_hierarchy) {
      /* remove self from the network */
      delself();
      if ((dump()) != 0) {
        debug(1,1,"Error in dumping network status, data may be lost.\n");
      }
    }
    debug(1, 1, "Waiting %d seconds for active connections to finish\n",
        shutdown_pending > 0 ? Config.lifetimeShutdown : 0);
#ifdef KILL_PARENT_OPT
    debug(1, 1, "Killing RunCache, pid %d\n", getppid());
    kill(getppid(), sig);
#endif
#if SA_RESETHAND == 0
    signal(SIGTERM, SIG_DFL);
    signal(SIGINT, SIG_DFL);
#endif
```

```
}

static void
serverConnectionsOpen(void)
{
    struct in_addr addr;
    struct sockaddr_in xaddr;
    u_short port;
    int len;
    int x;
    enter_suid();
    theHttpConnection = comm_open(SOCK_STREAM,
        0,
        Config.Addrs.tcp_incoming,
        Config.Port.http,
        COMM_NONBLOCKING,
        "HTTP Port");
    leave_suid();
    if (theHttpConnection < 0) {
        fatal("Cannot open HTTP Port");
    }
    fd_note(theHttpConnection, "HTTP socket");
    comm_listen(theHttpConnection);
    commSetSelect(theHttpConnection,
        COMM_SELECT_READ,
        asciiHandleConn,
        NULL, 0);
    debug(1, 1, "Accepting HTTP connections on FD %d.\n",
        theHttpConnection);

    if (!httpd_accel_mode || Config.Accel.withProxy) {
        if ((port = Config.Port.icp) > (u_short) 0) {
            enter_suid();
            theInIcpConnection = comm_open(SOCK_DGRAM,
                0,
                Config.Addrs.udp_incoming,
                port,
                COMM_NONBLOCKING,
                "ICP Port");
            leave_suid();
            if (theInIcpConnection < 0)
                fatal("Cannot open ICP Port");
            fd_note(theInIcpConnection, "ICP socket");
            commSetSelect(theInIcpConnection,
                COMM_SELECT_READ,
                icpHandleUdp,
                NULL, 0);
```

```
        comm_join_mcast_groups(theInIcpConnection);
        debug(1, 1, "Accepting ICP connections on FD %d.\n",
            theInIcpConnection);

        if ((addr = Config.Addrs.udp_outgoing).s_addr != inaddr_none) {
            enter_suid();
            theOutIcpConnection = comm_open(SOCK_DGRAM,
                0,
                addr,
                port,
                COMM_NONBLOCKING,
                "ICP Port");
            leave_suid();
            if (theOutIcpConnection < 0)
                fatal("Cannot open Outgoing ICP Port");
            commSetSelect(theOutIcpConnection,
                COMM_SELECT_READ,
                icpHandleUdp,
                NULL, 0);
            debug(1, 1, "Accepting ICP connections on FD %d.\n",
                theOutIcpConnection);
            fd_note(theOutIcpConnection, "Outgoing ICP socket");
            fd_note(theInIcpConnection, "Incoming ICP socket");
        } else {
            theOutIcpConnection = theInIcpConnection;
        }
        memset(&theOutICPAddr, '\0', sizeof(struct in_addr));
        len = sizeof(struct sockaddr_in);
        memset(&xaddr, '\0', len);
        x = getsockname(theOutIcpConnection,
            (struct sockaddr *) &xaddr, &len);
        if (x < 0)
            debug(50, 1, "theOutIcpConnection FD %d:  getsockname:  %s\n",
                theOutIcpConnection, xstrerror());
        else
            theOutICPAddr = xaddr.sin_addr;
    }
}
if (Config.vizHack.port) {
    vizSock = comm_open(SOCK_DGRAM,
        0,
        any_addr,
        0,
        COMM_NONBLOCKING,
        "VizHack Port");
    if (vizSock < 0)
        fatal("Could not open Viz Socket");
```

```
#if defined(IP_ADD_MEMBERSHIP) && defined(IP_MULTICAST_TTL)
        if (Config.vizHack.addr.s_addr > inet_addr("224.0.0.0")) {
            struct ip_mreq mr;
            char ttl = (char) Config.vizHack.mcast_ttl;
            memset((char *) &mr, '\0', sizeof(struct ip_mreq));
            mr.imr_multiaddr.s_addr = Config.vizHack.addr.s_addr;
            mr.imr_interface.s_addr = INADDR_ANY;
            x = setsockopt(vizSock,
                    IPPROTO_IP,
                    IP_ADD_MEMBERSHIP,
                    (char *) &mr,
                    sizeof(struct ip_mreq));
            if (x < 0)
                    debug(50, 1, "IP_ADD_MEMBERSHIP: FD %d, addr %s:  %s\n",
                        vizSock, inet_ntoa(Config.vizHack.addr), xstrerror());
            x = setsockopt(vizSock,
                    IPPROTO_IP,
                    IP_MULTICAST_TTL,
                    &ttl,
                    sizeof(char));
            if (x < 0)
                    debug(50, 1, "IP_MULTICAST_TTL: FD %d, TTL %d:  %s\n",
                        vizSock, Config.vizHack.mcast_ttl, xstrerror());
            ttl = 0;
            x = sizeof(char);
            getsockopt(vizSock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, &x);
            debug(1, 0, "vizSock on FD %d, ttl=%d\n", vizSock, (int) ttl);
        }
#else
        debug(1, 0, "vizSock:  Could not join multicast group\n");
#endif
        memset((char *) &Config.vizHack.S, '\0', sizeof(struct sockaddr_in));
        Config.vizHack.S.sin_family = AF_INET;
        Config.vizHack.S.sin_addr = Config.vizHack.addr;
        Config.vizHack.S.sin_port = htons(Config.vizHack.port);
    }
    clientdbInit();
    icmpOpen();
    netdbInit();
}

void
serverConnectionsClose(void)
{
    /* NOTE, this function will be called repeatedly while shutdown
     * is pending */
    if (theHttpConnection >= 0) {
```

```
        debug(1, 1, "FD %d Closing HTTP connection\n",
            theHttpConnection);
        comm_close(theHttpConnection);
        commSetSelect(theHttpConnection,
            COMM_SELECT_READ,
            NULL,
            NULL, 0);
        theHttpConnection = -1;
    }
    if (theInIcpConnection >= 0) {
        /* NOTE, don't close outgoing ICP connection, we need to write to
         * it during shutdown */
        debug(1, 1, "FD %d Closing ICP connection\n",
            theInIcpConnection);
        if (theInIcpConnection != theOutIcpConnection)
            comm_close(theInIcpConnection);
        commSetSelect(theInIcpConnection,
            COMM_SELECT_READ,
            NULL,
            NULL, 0);
        if (theInIcpConnection != theOutIcpConnection)
            commSetSelect(theOutIcpConnection,
                    COMM_SELECT_READ,
                    NULL,
                    NULL, 0);
        theInIcpConnection = -1;
    }
    if (icmp_sock > -1)
        icmpClose();
}

static void
mainReinitialize(void)
{
    debug(1, 0, "Restarting Squid Cache (version %s)...\n", version_string);
    /* Already called serverConnectionsClose and ipcacheShutdownServers() */
    neighborsDestroy();
    parseConfigFile(ConfigFile);
    _db_init(Config.Log.log, Config.debugOptions);
    dnsOpenServers();
    redirectOpenServers();
    serverConnectionsOpen();
    (void) ftpInitialize();
    if (theOutIcpConnection >= 0 && (!httpd_accel_mode || Config.Accel.withProxy))
        neighbors_open(theOutIcpConnection);
    debug(1, 0, "Ready to serve requests.\n");
}
```

```
static void
mainInitialize(void)
{
    static int first_time = 1;
    if (opt_catch_signals) {
        squid_signal(SIGSEGV, death, SA_NODEFER | SA_RESETHAND);
        squid_signal(SIGBUS, death, SA_NODEFER | SA_RESETHAND);
    }
    squid_signal(SIGPIPE, SIG_IGN, SA_RESTART);
    squid_signal(SIGCHLD, sig_child, SA_NODEFER | SA_RESTART);

    if (ConfigFile == NULL)
        ConfigFile = xstrdup(DefaultConfigFile);
    parseConfigFile(ConfigFile);

    leave_suid();               /* Run as non privilegied user */

    if (httpPortNumOverride != 1)
        Config.Port.http = (u_short) httpPortNumOverride;
    if (icpPortNumOverride != 1)
        Config.Port.icp = (u_short) icpPortNumOverride;

    _db_init(Config.Log.log, Config.debugOptions);
    fdstat_open(fileno(debug_log), FD_LOG);
    fd_note(fileno(debug_log), Config.Log.log);

    debug(1, 0, "Starting Squid Cache version %s for %s...\n",
        version_string,
        CONFIG_HOST_TYPE);
    debug(1, 1, "With %d file descriptors available\n", SQUID_MAXFD);

    if (first_time) {
        stmemInit();            /* stmem must go before at least redirect */
        disk_init();            /* disk_init must go before ipcache_init() */
    }
    ipcache_init();
    fqdncache_init();
    dnsOpenServers();
    redirectOpenServers();
    useragentOpenLog();
    (void) ftpInitialize();

#if MALLOC_DBG
    malloc_debug(0, malloc_debug_level);
#endif
```

```
    if (first_time) {
        /* module initialization */
        urlInitialize();
        stat_init(&HTTPCacheInfo, Config.Log.access);
        stat_init(&ICPCacheInfo, NULL);
        storeInit();

        if (Config.effectiveUser) {
            /* we were probably started as root, so cd to a swap
             * directory in case we dump core */
            if (chdir(swappath(0)) < 0) {
                debug(50, 0, "%s:  %s\n", swappath(0), xstrerror());
                fatal_dump("Cannot cd to swap directory?");
            }
        }
        /* after this point we want to see the mallinfo() output */
        do_mallinfo = 1;
    }
    serverConnectionsOpen();
    if (theOutIcpConnection >= 0 && (!httpd_accel_mode || Config.Accel.withProxy))
        neighbors_open(theOutIcpConnection);

    if (first_time)
        writePidFile();         /* write PID file */

    squid_signal(SIGUSR1, rotate_logs, SA_RESTART);
    squid_signal(SIGUSR2, sigusr2_handle, SA_RESTART);
    squid_signal(SIGHUP, reconfigure, SA_RESTART);
    squid_signal(SIGTERM, shut_down, SA_NODEFER | SA_RESETHAND | SA_RESTART);
    squid_signal(SIGINT, shut_down, SA_NODEFER | SA_RESETHAND | SA_RESTART);
    debug(1, 0, "Ready to serve requests.\n");

    if (first_time) {
        eventAdd("storePurgeOld", storePurgeOld, NULL, Config.cleanRate);
        eventAdd("storeMaintain", storeMaintainSwapSpace, NULL, 1);
        eventAdd("storeDirClean", storeDirClean, NULL, 15);
        if (Config.Announce.on)
            eventAdd("send_announce", send_announce, NULL, 3600);
        eventAdd("ipcache_purgelru", (EVH) ipcache_purgelru, NULL, 10);
    }
    first_time = 0;
}

int
main(int argc, char **argv)
{
    int errcount = 0;
```

```
    int n;                          /* # of GC'd objects */
    time_t loop_delay;
    /* begin DH changes */
    time_t ltime, ctime;
    /* end DH changes */

    /* call mallopt() before anything else */
#if HAVE_MALLOPT
#ifdef M_GRAIN
    /* Round up all sizes to a multiple of this */
    mallopt(M_GRAIN, 16);
#endif
#ifdef M_MXFAST
    /* biggest size that is considered a small block */
    mallopt(M_MXFAST, 256);
#endif
#ifdef M_NBLKS
    /* allocate this many small blocks at once */
    mallopt(M_NLBLKS, 32);
#endif
#endif /* HAVE_MALLOPT */

    memset(&local_addr, '\0', sizeof(struct in_addr));
    local_addr.s_addr = inet_addr(localhost);
    memset(&any_addr, '\0', sizeof(struct in_addr));
    any_addr.s_addr = inet_addr("0.0.0.0");
    memset(&no_addr, '\0', sizeof(struct in_addr));
    no_addr.s_addr = inet_addr("255.255.255.255");
    inaddr_none = inet_addr("X");

    errorInitialize();

    squid_starttime = getCurrentTime();
    failure_notify = fatal_dump;

    mainParseOptions(argc, argv);

    /* send signal to running copy and exit */
    if (opt_send_signal != -1) {
        sendSignal();
        /* NOTREACHED */
    }
    setMaxFD();

    if (opt_catch_signals)
        for (n = SQUID_MAXFD; n > 2; n--)
            close(n);
```

```
/*init comm module */
comm_init();

/* we have to init fdstat here. */
fdstat_init(PREOPEN_FD);
fdstat_open(0, FD_LOG);
fdstat_open(1, FD_LOG);
fdstat_open(2, FD_LOG);
fd_note(0, "STDIN");
fd_note(1, "STDOUT");
fd_note(2, "STDERR");

/* preinit for debug module */
debug_log = stderr;
hash_init(0);

mainInitialize();

/* begin DH changes */
if (opt_dynamic_hierarchy) {
  const char *myhostname = getMyHostname();
  if ((addself()) != 0) {
        debug(1,0, "Couldn't join network.\n");
  }
  ltime = squid_starttime;
  /* get the listing */
  if ((getnet()) != 0) {
        debug(1, 0, "Error in getting new network status.\n");
  }
  num_hosts = dhBuildUI();
}
/* end DH changes */

/* main loop */
for (;;) {
  if (rotate_pending) {
        ftpServerClose();
        icmpClose();
        _db_rotate_log();           /* cache.log */
        storeWriteCleanLog();
        storeRotateLog();           /* store.log */
        stat_rotate_log();          /* access.log */
        useragentRotateLog();       /* useragent.log */
        (void) ftpInitialize();
        icmpOpen();
        rotate_pending = 0;
```

```
  }
  eventRun();
  if ((loop_delay = eventNextTime()) < 0)
      loop_delay = 0;

  /* begin DH changes */
  ctime = getCurrentTime();
  if ((ctime - ltime)>= UPDATE_TIME) {
      getupdate(ltime);
      ltime = ctime;
      num_hosts = dhBuildUI();
  }
  /* end DH changes*/

  switch (comm_select(loop_delay)) {
  case COMM_OK:
      errcount = 0;     /* reset if successful */
      break;
  case COMM_ERROR:
      errcount++;
      debug(1, 0, "Select loop Error.  Retry %d\n", errcount);
      if (errcount == 10)
        fatal_dump("Select Loop failed!");
      break;
  case COMM_SHUTDOWN:
      /* delayed close so we can transmit while shutdown pending */
      if (theOutIcpConnection > 0) {
        comm_close(theOutIcpConnection);
        theOutIcpConnection = -1;
      }
      if (shutdown_pending) {
        normal_shutdown();
        exit(0);
      } else if (reread_pending) {
        mainReinitialize();
        reread_pending = 0;     /* reset */
      } else {
        fatal_dump("MAIN: SHUTDOWN from comm_select, but nothing pending.");
      }
      break;
  case COMM_TIMEOUT:
      break;
  default:
      fatal_dump("MAIN: Internal error -- this should never happen.");
      break;
  }
}
```

```
    /* NOTREACHED */
    return 0;
}

static void
sendSignal(void)
{
    pid_t pid;
    debug_log = stderr;
    if (ConfigFile == NULL)
        ConfigFile = xstrdup(DefaultConfigFile);
    parseConfigFile(ConfigFile);
    pid = readPidFile();
    if (pid > 1) {
        if (kill(pid, opt_send_signal) &&
        /* ignore permissions if just running check */
            !(opt_send_signal == 0 && errno == EPERM)) {
            fprintf(stderr, "%s: ERROR: Could not send ", appname);
            fprintf(stderr, "signal %d to process %d: %s\n",
                opt_send_signal, (int) pid, xstrerror());
            exit(1);
        }
    } else {
        fprintf(stderr, "%s: ERROR: No running copy\n", appname);
        exit(1);
    }
    /* signal successfully sent */
    exit(0);
}
```

B.3 proto.c

```
/*
 * $Id: proto.c,v 1.3 1997/01/08 04:56:52 marthag Exp $
 *
 * DEBUG: section 17     Neighbor Selection
 * AUTHOR: Harvest Derived
 *
 * SQUID Internet Object Cache  http://squid.nlanr.net/Squid/
 * --------------------------------------------------------------
 *
 *  Squid is the result of efforts by numerous individuals from the
 *  Internet community.  Development is led by Duane Wessels of the
 *  National Laboratory for Applied Network Research and funded by
 *  the National Science Foundation.
 *
```

```
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */

/*
 * Copyright (c) 1994, 1995.  All rights reserved.
 *
 *   The Harvest software was developed by the Internet Research Task
 *   Force Research Group on Resource Discovery (IRTF-RD):
 *
 *         Mic Bowman of Transarc Corporation.
 *         Peter Danzig of the University of Southern California.
 *         Darren R. Hardy of the University of Colorado at Boulder.
 *         Udi Manber of the University of Arizona.
 *         Michael F. Schwartz of the University of Colorado at Boulder.
 *         Duane Wessels of the University of Colorado at Boulder.
 *
 *   This copyright notice applies to software in the Harvest
 *   "src/" directory only.  Users should consult the individual
 *   copyright notices in the "components/" subdirectories for
 *   copyright information about other software bundled with the
 *   Harvest source code distribution.
 *
 * TERMS OF USE
 *
 *   The Harvest software may be used and re-distributed without
 *   charge, provided that the software origin and research team are
 *   cited in any use of the system.  Most commonly this is
 *   accomplished by including a link to the Harvest Home Page
 *   (http://harvest.cs.colorado.edu/) from the query page of any
 *   Broker you deploy, as well as in the query result pages.  These
 *   links are generated automatically by the standard Broker
 *   software distribution.
 *
 *   The Harvest software is provided "as is", without express or
```

```
 *   implied warranty, and with no support nor obligation to assist
 *   in its use, correction, modification or enhancement.  We assume
 *   no liability with respect to the infringement of copyrights,
 *   trade secrets, or any patents, and are not responsible for
 *   consequential damages.  Proper use of the Harvest software is
 *   entirely the responsibility of the user.
 *
 * DERIVATIVE WORKS
 *
 *   Users may make derivative works from the Harvest software, subject
 *   to the following constraints:
 *
 *     - You must include the above copyright notice and these
 *       accompanying paragraphs in all forms of derivative works,
 *       and any documentation and other materials related to such
 *       distribution and use acknowledge that the software was
 *       developed at the above institutions.
 *
 *     - You must notify IRTF-RD regarding your distribution of
 *       the derivative work.
 *
 *     - You must clearly notify users that your are distributing
 *       a modified version and not the original Harvest software.
 *
 *     - Any derivative product is also subject to these copyright
 *       and use restrictions.
 *
 *   Note that the Harvest software is NOT in the public domain.  We
 *   retain copyright, as specified above.
 *
 * HISTORY OF FREE SOFTWARE STATUS
 *
 *   Originally we required sites to license the software in cases
 *   where they were going to build commercial products/services
 *   around Harvest.  In June 1995 we changed this policy.  We now
 *   allow people to use the core Harvest software (the code found in
 *   the Harvest "src/" directory) for free.  We made this change
 *   in the interest of encouraging the widest possible deployment of
 *   the technology.  The Harvest software is really a reference
 *   implementation of a set of protocols and formats, some of which
 *   we intend to standardize.  We encourage commercial
 *   re-implementations of code complying to this set of standards.
 */

#include "squid.h"
```

```
static int matchLocalDomain _PARAMS((const char *));
static int protoCantFetchObject _PARAMS((int, StoreEntry *, char *));
static int protoNotImplemented _PARAMS((int, const char *, StoreEntry *));
static int protoDNSError _PARAMS((int, StoreEntry *));
static void protoDataFree _PARAMS((int, protodispatch_data *));
static void protoDispatchDNSHandle _PARAMS((int, const ipcache_addrs *, void *));

#define OUTSIDE_FIREWALL 0
#define INSIDE_FIREWALL  1
#define NO_FIREWALL      2

/* for debugging */
static char *firewall_desc_str[] =
{
    "OUTSIDE_FIREWALL",
    "INSIDE_FIREWALL",
    "NO_FIREWALL"
};

char *IcpOpcodeStr[] =
{
    "ICP_INVALID",
    "ICP_QUERY",
    "ICP_HIT",
    "ICP_MISS",
    "ICP_ERR",
    "ICP_SEND",
    "ICP_SENDA",
    "ICP_DATABEG",
    "ICP_DATA",
    "ICP_DATAEND",
    "ICP_SECHO",
    "ICP_DECHO",
    "ICP_OP_UNUSED0",
    "ICP_OP_UNUSED1",
    "ICP_OP_UNUSED2",
    "ICP_OP_UNUSED3",
    "ICP_OP_UNUSED4",
    "ICP_OP_UNUSED5",
    "ICP_OP_UNUSED6",
    "ICP_OP_UNUSED7",
    "ICP_OP_UNUSED8",
    "ICP_RELOADING",        /* access denied while store is reloading */
    "ICP_DENIED",
    "ICP_HIT_OBJ",
    "ICP_END"
};
```

```
#if DELAY_HACK
extern int _delay_fetch;
#endif

static void
protoDataFree(int fdunused, protodispatch_data * protoData)
{
    requestUnlink(protoData->request);
    safe_free(protoData);
}

/* called when DNS lookup is done by ipcache. */
static void
protoDispatchDNSHandle(int unused1, const ipcache_addrs * ia, void *data)
{
    edge *e = NULL;
    struct in_addr srv_addr;
    protodispatch_data *protoData = (protodispatch_data *) data;
    StoreEntry *entry = protoData->entry;
    request_t *req = protoData->request;

    /* NOTE: We get here after a DNS lookup, whether or not the
     * lookup was successful.  Even if the URL hostname is bad,
     * we might still ping the hierarchy */

    BIT_RESET(entry->flag, IP_LOOKUP_PENDING);

    if (protoData->direct_fetch == DIRECT_YES) {
        if (ia == NULL) {
            protoDNSError(protoData->fd, entry);
            return;
        }
        hierarchyNote(req, HIER_DIRECT, 0, req->host);
        protoStart(protoData->fd, entry, NULL, req);
        return;
    }
    if (protoData->direct_fetch == DIRECT_MAYBE && (Config.local_ip_list || Config.firewall_ip_list)) {
        if (ia == NULL) {
            debug(17, 3, "Unknown host:  %s\n", req->host);
            protoData->direct_fetch = DIRECT_NO;
        } else if (Config.firewall_ip_list) {
            srv_addr = ia->in_addrs[ia->cur];
            if (ip_access_check(srv_addr, Config.firewall_ip_list) == IP_DENY) {
                hierarchyNote(req, HIER_LOCAL_IP_DIRECT, 0, req->host);
                protoStart(protoData->fd, entry, NULL, req);
                return;
```

```
            } else {
                protoData->direct_fetch = DIRECT_NO;
            }
        } else if (Config.local_ip_list) {
            srv_addr = ia->in_addrs[ia->cur];
            if (ip_access_check(srv_addr, Config.local_ip_list) == IP_DENY) {
                hierarchyNote(req, HIER_LOCAL_IP_DIRECT, 0, req->host);
                protoStart(protoData->fd, entry, NULL, req);
                return;
            }
        }
    }
    if ((e = protoData->single_parent) &&
        (Config.singleParentBypass || protoData->direct_fetch == DIRECT_NO)) {
        /* Only one parent for this host, and okay to skip pinging stuff */
        hierarchyNote(req, HIER_SINGLE_PARENT, 0, e->host);
        protoStart(protoData->fd, entry, e, req);
        return;
    }
    if (protoData->n_edges == 0 && protoData->direct_fetch == DIRECT_NO) {
        if ((e = protoData->default_parent)) {
            hierarchyNote(req, HIER_DEFAULT_PARENT, 0, e->host);
            protoStart(protoData->fd, entry, e, req);
        } else {
            hierarchyNote(req, HIER_NO_DIRECT_FAIL, 0, req->host);
            protoCantFetchObject(protoData->fd, entry, "No neighbors or parents to query and the host is beyond your firewall.");
        }
        return;
    }
    if (!neighbors_do_private_keys && !protoData->query_neighbors &&
        (e = getFirstUpParent(req))) {
        /* for private objects we should just fetch directly (because
         * icpHandleUdp() won't properly deal with the ICP replies). */
        hierarchyNote(req, HIER_FIRSTUP_PARENT, 0, e->host);
        protoStart(protoData->fd, entry, e, req);
        return;
    } else if (protoData->direct_fetch == DIRECT_MAYBE && ia
        && netdbHops(ia->in_addrs[ia->cur]) <= Config.minDirectHops) {
        hierarchyNote(req, HIER_DIRECT, 0, req->host);
        protoStart(protoData->fd, entry, NULL, req);
        return;
    } else if (neighborsUdpPing(protoData)) {
        /* call neighborUdpPing and start timeout routine */
        if (entry->ping_status != PING_NONE)
            fatal_dump("protoDispatchDNSHandle:  bad ping_status");
        if (entry->store_status != STORE_PENDING)
            fatal_dump("protoDispatchDNSHandle:  bad store_status");
```

```
        if (entry->swap_status != NO_SWAP)
            fatal_dump("protoDispatchDNSHandle:  bad swap_status");
        entry->ping_status = PING_WAITING;
        commSetSelect(protoData->fd,
            COMM_SELECT_TIMEOUT,
            (PF) getFromDefaultSource,
            (void *) entry,
            Config.neighborTimeout);
#ifdef DELAY_HACK
        if (protoData->delay_fetch && entry->mem_obj)
            entry->mem_obj->e_pings_n_pings++;
#endif
        return;
    }
    if ((e = protoData->default_parent)) {
        hierarchyNote(req, HIER_DEFAULT_PARENT, protoData->fd, e->host);
        protoStart(protoData->fd, entry, e, req);
    } else if (protoData->direct_fetch == DIRECT_NO) {
        hierarchyNote(req, HIER_NO_DIRECT_FAIL, 0, req->host);
        protoCantFetchObject(protoData->fd, entry,
            "No neighbors or parents were queried and the host is beyond your firewall.");
    } else if (ia == NULL) {
        protoDNSError(protoData->fd, entry);
    } else {
        hierarchyNote(req, HIER_DIRECT, 0, req->host);
        protoStart(protoData->fd, entry, NULL, req);
    }
}

int
protoDispatch(int fd, char *url, StoreEntry * entry, request_t * request)
{
    protodispatch_data *protoData = NULL;
    const char *method;
    char *request_hdr;

    method = RequestMethodStr[request->method];
    request_hdr = entry->mem_obj->mime_hdr;

    debug(17, 5, "protoDispatch:  %s URL: %s\n", method, url);
    debug(17, 10, "request_hdr:  %s\n", request_hdr);

    if (request->protocol == PROTO_CACHEOBJ)
        return protoStart(fd, entry, NULL, request);
    if (request->protocol == PROTO_WAIS)
        return protoStart(fd, entry, NULL, request);
```

```
  protoData = xcalloc(1, sizeof(protodispatch_data));
  protoData->fd = fd;
  protoData->url = url;
  protoData->entry = entry;
  protoData->request = requestLink(request);
  entry->mem_obj->request = requestLink(request);
  comm_add_close_handler(fd,
          (PF) protoDataFree,
          (void *) protoData);

  protoData->inside_firewall = matchInsideFirewall(request->host);
  protoData->query_neighbors = BIT_TEST(entry->flag, HIERARCHICAL);
  /* begin DH changes */
  if (!opt_dynamic_hierarchy) {
    protoData->single_parent = getSingleParent(request);
    protoData->default_parent = getDefaultParent(request);
    protoData->n_edges = neighborsCount(request);
  }
  else {
    struct _edge * dhparent = dhGetParent(url);
    protoData->single_parent = dhparent;
    protoData->default_parent = dhparent;
    protoData->n_edges = dhNeighborsCount(dhparent);
  }
  /* end DH changes */
#ifdef DELAY_HACK
  protoData->delay_fetch = _delay_fetch;
#endif

  debug(17, 2, "protoDispatch:  inside_firewall = %d (%s)\n",
          protoData->inside_firewall,
          firewall_desc_str[protoData->inside_firewall]);
  debug(17, 2, "protoDispatch:  query_neighbors = %d\n", protoData->query_neighbors);
  debug(17, 2, "protoDispatch:                  n_edges = %d\n", protoData->n_edges);
  debug(17, 2, "protoDispatch:      single_parent = %s\n",
          protoData->single_parent ? protoData->single_parent->host : "N/A");
  debug(17, 2, "protoDispatch:    default_parent = %s\n",
          protoData->default_parent ? protoData->default_parent->host : "N/A");

  if (Config.firewall_ip_list) {
          /* Have to look up the url address so we can compare it */
          protoData->source_ping = Config.sourcePing;
          protoData->direct_fetch = DIRECT_MAYBE;
          BIT_SET(entry->flag, IP_LOOKUP_PENDING);
          ipcache_nbgethostbyname(request->host,
              fd,
              protoDispatchDNSHandle,
```

```
            (void *) protoData);
    } else if (!protoData->inside_firewall) {
        /* There are firewall restrictsions, and this host is outside. */
        /* No DNS lookups, call protoDispatchDNSHandle() directly */
        protoData->source_ping = 0;
        protoData->direct_fetch = DIRECT_NO;
        protoDispatchDNSHandle(fd,
            NULL,
            (void *) protoData);
    } else if (matchLocalDomain(request->host) || !protoData->query_neighbors) {
        /* will fetch from source */
        protoData->direct_fetch = DIRECT_YES;
        BIT_SET(entry->flag, IP_LOOKUP_PENDING);
        ipcache_nbgethostbyname(request->host,
            fd,
            protoDispatchDNSHandle,
            (void *) protoData);
    } else if (protoData->n_edges == 0) {
        /* will fetch from source */
        protoData->direct_fetch = DIRECT_YES;
        BIT_SET(entry->flag, IP_LOOKUP_PENDING);
        ipcache_nbgethostbyname(request->host,
            fd,
            protoDispatchDNSHandle,
            (void *) protoData);
    } else if (Config.local_ip_list) {
        /* Have to look up the url address so we can compare it */
        protoData->source_ping = Config.sourcePing;
        protoData->direct_fetch = DIRECT_MAYBE;
        BIT_SET(entry->flag, IP_LOOKUP_PENDING);
        ipcache_nbgethostbyname(request->host,
            fd,
            protoDispatchDNSHandle,
            (void *) protoData);
    } else if (protoData->single_parent && Config.singleParentBypass &&
        !(protoData->source_ping = Config.sourcePing)) {
        /* will fetch from single parent */
        protoData->direct_fetch = DIRECT_MAYBE;
        protoDispatchDNSHandle(fd,
            NULL,
            (void *) protoData);
    } else {
        /* will use ping resolution */
        protoData->source_ping = Config.sourcePing;
        protoData->direct_fetch = DIRECT_MAYBE;
        BIT_SET(entry->flag, IP_LOOKUP_PENDING);
        ipcache_nbgethostbyname(request->host,
```

```
                fd,
                protoDispatchDNSHandle,
                (void *) protoData);
    }
    return 0;
}

int
protoUnregister(int fd, StoreEntry * entry, request_t * request, struct in_addr src_addr)
{
    char *url = entry ? entry->url : NULL;
    char *host = request ? request->host : NULL;
    protocol_t proto = request ? request->protocol : PROTO_NONE;
    debug(17, 5, "protoUnregister FD %d '%s'\n", fd, url ? url : "NULL");
    if (proto == PROTO_CACHEOBJ)
        return 0;
    if (url)
        redirectUnregister(url, fd);
    if (src_addr.s_addr != inaddr_none)
        fqdncacheUnregister(src_addr, fd);
    if (host)
        ipcache_unregister(host, fd);
    if (entry == NULL)
        return 0;
    if (BIT_TEST(entry->flag, ENTRY_DISPATCHED))
        return 0;
    if (entry->mem_status != NOT_IN_MEMORY)
        return 0;
    if (entry->store_status != STORE_PENDING)
        return 0;
    protoCancelTimeout(fd, entry);
    squid_error_entry(entry, ERR_CLIENT_ABORT, NULL);
    return 1;
}

void
protoCancelTimeout(int fd, StoreEntry * entry)
{
    /* If fd = 0 then this thread was called from neighborsUdpAck and
     * we must look up the FD in the pending list. */
    if (!fd)
        fd = storeFirstClientFD(entry->mem_obj);
    if (fd < 1) {
        debug(17, 1, "protoCancelTimeout:  No client for '%s'\n", entry->url);
        return;
    }
    debug(17, 2, "protoCancelTimeout:  FD %d '%s'\n", fd, entry->url);
```

```
    if (fdstatGetType(fd) != FD_SOCKET) {
        debug_trap("protoCancelTimeout:  called on non-socket");
        return;
    }
    /* cancel the timeout handler */
    commSetSelect(fd,
        COMM_SELECT_TIMEOUT,
        NULL,
        NULL,
        0);
}

/*
 *  Called from comm_select() if neighbor pings timeout
 *  or from neighborsUdpAck() if all neighbors miss.
 */
int
getFromDefaultSource(int fd, StoreEntry * entry)
{
    edge *e = NULL;
    char *url = NULL;
    request_t *request = entry->mem_obj->request;

    url = entry->url;

    /* if fd != 0 then we were called from comm_select() because the
     * timeout occured.  Otherwise we were called from neighborsUdpAck(). */

    if (fd) {
        entry->ping_status = PING_TIMEOUT;
        debug(17, 5, "getFromDefaultSource:  Timeout occured pinging for '%s'\n",
            url);
    }
    /* Check if someone forgot to disable the read timer */
    if (BIT_TEST(entry->flag, ENTRY_DISPATCHED))
        fatal_dump("getFromDefaultSource:  object already being fetched");
    if ((e = entry->mem_obj->e_pings_first_miss)) {
        hierarchyNote(request, HIER_FIRST_PARENT_MISS, fd, e->host);
        return protoStart(fd, entry, e, request);
    }
    if (matchInsideFirewall(request->host)) {
        if (ipcache_gethostbyname(request->host, 0) == NULL)
            return protoDNSError(fd, entry);
        hierarchyNote(request, HIER_DIRECT, fd, request->host);
        return protoStart(fd, entry, NULL, request);
    }
    if ((e = getDefaultParent(request))) {
```

```
        hierarchyNote(request, HIER_DEFAULT_PARENT, fd, e->host);
        return protoStart(fd, entry, e, request);
    }
    if ((e = getSingleParent(request))) {
        hierarchyNote(request, HIER_SINGLE_PARENT, fd, e->host);
        return protoStart(fd, entry, e, request);
    }
    if ((e = getFirstUpParent(request))) {
        hierarchyNote(request, HIER_FIRSTUP_PARENT, fd, e->host);
        return protoStart(fd, entry, e, request);
    }
    hierarchyNote(request, HIER_NO_DIRECT_FAIL, fd, request->host);
    protoCancelTimeout(fd, entry);
    protoCantFetchObject(fd, entry,
        "No ICP replies received and the host is beyond the firewall.");
    return 0;
}

int
protoStart(int fd, StoreEntry * entry, edge * e, request_t * request)
{
    char *url = entry->url;
    char *request_hdr = entry->mem_obj->mime_hdr;
    int request_hdr_sz = entry->mem_obj->mime_hdr_sz;
    debug(17, 5, "protoStart:  FD %d:  Fetching '%s %s' from %s\n",
        fd,
        RequestMethodStr[request->method],
        entry->url,
        e ? e->host : "source");
    if (BIT_TEST(entry->flag, ENTRY_DISPATCHED))
        fatal_dump("protoStart:  object already being fetched");
    BIT_SET(entry->flag, ENTRY_DISPATCHED);
    protoCancelTimeout(fd, entry);
    netdbPingSite(request->host);
    if (e) {
        e->stats.fetches++;
        return proxyhttpStart(url, request, entry, e);
    } else if (request->protocol == PROTO_HTTP) {
        return httpStart(url, request, request_hdr, request_hdr_sz, entry);
    } else if (request->protocol == PROTO_GOPHER) {
        return gopherStart(fd, url, entry);
    } else if (request->protocol == PROTO_FTP) {
        return ftpStart(fd, url, request, entry);
    } else if (request->protocol == PROTO_WAIS) {
        return waisStart(fd, url, request->method, request_hdr, entry);
    } else if (request->protocol == PROTO_CACHEOBJ) {
        return objcacheStart(fd, url, entry);
```

```
    } else if (request->method == METHOD_CONNECT) {
        fatal_dump("protoStart() should not be handling CONNECT");
        return 0;
    } else {
        return protoNotImplemented(fd, url, entry);
    }
    /* NOTREACHED */
}


static int
protoNotImplemented(int fd, const char *url, StoreEntry * entry)
{
    LOCAL_ARRAY(char, buf, 256);

    debug(17, 1, "protoNotImplemented:  Cannot retrieve '%s'\n", url);

    buf[0] = '\0';
    if (httpd_accel_mode)
        strcpy(buf, "cached is running in HTTPD accelerator mode, so it does not allow the normal URL syntax.");
    else
        sprintf(buf, "Your URL may be incorrect:  '%s'\n", url);

    squid_error_entry(entry, ERR_NOT_IMPLEMENTED, NULL);
    return 0;
}

static int
protoCantFetchObject(int fd, StoreEntry * entry, char *reason)
{
    LOCAL_ARRAY(char, buf, 2048);

    debug(17, 1, "protoCantFetchObject:  FD %d %s\n", fd, reason);
    debug(17, 1, "--> '%s'\n", entry->url);

    buf[0] = '\0';
    sprintf(buf, "%s\n\nThe cache administrator may need to double-check the cache configuration.",
        reason);
    squid_error_entry(entry, ERR_CANNOT_FETCH, buf);
    return 0;
}

static int
protoDNSError(int fd, StoreEntry * entry)
{
    debug(17, 2, "protoDNSError:  FD %d '%s'\n", fd, entry->url);
    protoCancelTimeout(fd, entry);
```

```
    squid_error_entry(entry, ERR_DNS_FAIL, dns_error_message);
    return 0;
}

/*
 * return 0 if the host is outside the firewall (no domains matched), and
 * return 1 if the host is inside the firewall or no domains at all.
 */
int
matchInsideFirewall(const char *host)
{
    const wordlist *s = Config.inside_firewall_list;
    const char *key = NULL;
    int result = NO_FIREWALL;
    struct in_addr addr;
    if (!s)
        /* no domains, all hosts are "inside" the firewall */
        return NO_FIREWALL;
    for (; s; s = s->next) {
        key = s->key;
        if (!strcasecmp(key, "none"))
            /* no domains are inside the firewall, all domains are outside */
            return OUTSIDE_FIREWALL;
        if (*key == '!') {
            key++;
            result = OUTSIDE_FIREWALL;
        } else {
            result = INSIDE_FIREWALL;
        }
        if (matchDomainName(key, host))
            return result;
    }
    /* Check for dotted-quads */
    if (Config.firewall_ip_list) {
        if ((addr.s_addr = inet_addr(host)) != inaddr_none) {
            if (ip_access_check(addr, Config.firewall_ip_list) == IP_DENY)
                return INSIDE_FIREWALL;
        }
    }
    /* all through the list and no domains matched, this host must
     * not be inside the firewall, it must be outside */
    return OUTSIDE_FIREWALL;
}

static int
matchLocalDomain(const char *host)
{
```

```
    const wordlist *s = NULL;
    for (s = Config.local_domain_list; s; s = s->next) {
        if (matchDomainName(s->key, host))
            return 1;
    }
    return 0;
}
```

References

[1] Bowman, C., Peter B. Danzig, Udi Manber and Michael F. Schwartz. *Scalable Internet Resource Discovery: Research Problems and Approaches.* Communications of the ACM, 37(8), pp. 98-107, August 1994.

[2] Chankhunthod, Anawat, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz and Kurt J. Worrell. *A Hierarchical Internet Object Cache.* Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, March 1995.

[3] Hardy, Darren R., Michael F. Schwartz, Duane Wessels. *Harvest User's Manual.* Available at `http://harvest.transarc.com/afs/transarc.com/public/trg/Harvest/user-manual/`

[4] Karger, David R., et al, *Relieving Hot Spots on the World Wide Web.* Available at `http://theory.lcs.mit.edu/~karger/Papers/web.ps.gz`

[5] Wessels, Duane. *A Distributed Testbed for National Information Provisioning.* Available at `http://www.nlanr.net/Cache/`

[6] Wessels, Duane. *Intelligent Caching for World-Wide Web Objects.* Interdisciplinary Telecommunications Program, University of Colorado - Boulder, 1995. M.S. Thesis.

[7] Wessels, Duane. *Internet Cache Protocol (ICP) version 2.* Available at `http://www.nlanr.net/Cache/ICP/ICP-id.txt`

[8] Wessels, Duane. *Squid Internet Object Cache.* Available at `http://www.nlanr.net/Squid/`