Project 2: Dynamic DNS, Part B

EE122: Introduction to Communication Networks (Fall 2008)
Department of Electrical Engineering and Computer Science
College of Engineering
University of California, Berkeley
Due Dec 12 @ 11:59:59 pm

1 Overview

In Part A of the project, you built a dynamic Domain Name System (DNS) server and client to install name → IP mappings in your server. In Part B, you will extend your DNS server to perform peer-to-peer queries with the other DNS servers to learn DNS records installed at those servers.

More specifically, you will construct and maintain a peer-to-peer network composed of your servers and your fellow students’ servers. When your server gets a query for a name it doesn’t know, it will perform a search within the P2P network to see if anyone else knows the name. All the communication in this part will use the DNS protocol’s message format. However, we will add two new Resource Record types within that format.

2 P2P network construction and maintenance

New message type

We begin by describing the message format changes. Later we will describe how this is actually used.

Your servers will communicate using a new Resource Record type, PEER. When it appears in the Question section, this type indicates that the sender desires to learn about a new peer. When it appears in the Answer section, it gives the address of a peer. The format is as follows. Recall that the Resource Record contains a DOMAIN NAME, TYPE, CLASS, TTL, RESOURCE DATA LENGTH, and RESOURCE DATA:

- DOMAIN NAME: for outgoing messages, your server should set the domain name to be any value which is unique to the instance of your server. The domain name that you use is not important except that it should be unique to the instance of the server. To do that you can use the following code, run once when your server starts up:

  ```c
  char myname[30]; sprintf(myname, "myusername-%d", time(NULL));
  ```

  For messages you receive, you may assume that there is something in the name field, but you don’t need to use this name at all. (The reason a non-empty name is included is that it is more convenient to have a non-empty name for the serialize_deserialize_dns_names code.)

- TYPE is set to the value 100 to indicate a PEER Resource Record.
- CLASS should be set to 1, corresponding to “Internet class”, as usual.
- TIME TO LIVE is, as usual, the number of remaining seconds before the record should expire.
- RESOURCE DATA contains a 4-byte IP address followed by a 2-byte port number. (So therefore you know what the RESOURCE DATA LENGTH is.) The IP address is encoded as in Part A. The port number is big endian.
Server operations

We will construct the P2P network with the help of a tracker, similar to BitTorrent, which will suggest other servers to which your server can connect. Each server will periodically send a keepalive message to its neighbors so they know it is still alive.

When your server starts up, it should now take 3 command-line arguments: the port to which it should bind, the tracker’s IP address, and the tracker’s port:

```bash
/server server_port tracker_ip tracker_port
```

As of November 26, the tracker is running on 169.229.50.36 at port 10000.

The server operates as follows:

- The server maintains a table of neighbors, containing their IP address, port, and a TTL. Initially, the tracker is the only entry. As in the existing table of names, entries are removed after their TTL expires, except that the tracker should never expire. (The tracker will not bother to send periodic keepalive messages to its neighbors.)

- **Upon startup** and periodically **every KEEPALIVE_INTERVAL seconds thereafter**, the server sends to each neighbor a keepalive message: a DNS Response message with a PEER Resource Record in the Answer section. In that RR, your server puts its own IP and port in the RESOURCE DATA. (What do you think the TTL should be set to, in order to maintain the network topology reliably as well as not keep around too many stale entries?)

- Upon receipt of a such a keepalive message with server X in the RESOURCE DATA, the server looks up X in its neighbor table. If X is not the server itself and is not present already, it creates a new entry. Otherwise it just refreshes the TTL for X in its neighbor table, based on the TTL in the keepalive message. (Note that your server does not immediately send any acknowledgement of this keepalive—but because X is now a neighbor, it will send a keepalive in the reverse direction soon.)

- **Upon startup** and periodically **every KEEPALIVE_INTERVAL seconds thereafter**, if the server has fewer than TARGET_NEIGHBORS neighbors in its table (counting the tracker as a neighbor), it sends to the tracker a DNS Query message with a QUESTION SECTION entry of type PEER. The tracker is expected to reply with a keepalive message with the IP and port set to a random one of the tracker’s neighbors; your server will handle receipt of this message as described in the previous step.

Use the following values for the constants mentioned above:

- **KEEPALIVE_INTERVAL** = 5
- **TARGET_NEIGHBORS** = 4

3 Querying the network

Basic algorithm

Whenever your server gets a query (i.e., a standard DNS query as in Part A) for a name that it doesn’t know, it will query the peer-to-peer network of DNS servers to see if anyone else knows the name. Similar to early versions of the P2P network Gnutella, the query will be flooded through the network, from the originator to its neighbors, to its neighbors’ neighbors, and so on.

Flooding a query to all nodes in a large network may consume significant resources. Early versions of Gnutella mitigated this problem by placing a limit on the maximum number of hops a query could travel away from the source. We will use a somewhat more sophisticated method: a packet limit. This value represents the maximum total number of packets that should be sent on behalf of a request, directly or indirectly. So if a query arrives with packet limit $p$ and the server forwards it to $n$ neighbors, then there are $p - n$ remaining packets that may be sent on behalf of this query. It allocates that budget among each of the $n$ neighbors, telling each one that it is allowed to send at most $\frac{p-n}{n}$ packets.
For example, consider the topology below. Node A sends a query to its two neighbors with \( p = 2 \). The graph edges are annotated with the value of \( p \) in each query message. Note that the query reaches D but not H, even though they are both 3 hops away from A.

\[
\begin{array}{cccc}
\text{A} & \text{B} & \text{C} & \text{D} \\
p=2 & p=1 & p=0 & \\
\text{E} & \text{F} & \text{G} & \text{H} \\
p=2 & p=0 & \text{(no msg)} & \\
\end{array}
\]

**New message type**

A peer-to-peer query message is just like a normal DNS Query message from Part A of the project, with the following differences:

- In the **QUESTION SECTION**, there is one entry (as usual), but the type is set to the value 101, which is our new P2PQUERY message type.

- In the **ANSWER SECTION**—which is empty in normal queries from Part A—we now have a single resource record whose fields are set as follows:
  
  - **DOMAIN NAME** is the name being queried, in the same format as in Part A of the project.
  - **TYPE** is set to the value 101 to indicate a P2PQUERY resource record.
  - **TIME TO LIVE** is used to store the **packet limit** as described above.
  - **CLASS** should be set to 1, corresponding to “Internet class”, as usual.
  - **RESOURCE DATA** contains the **reply-to address**: an IP address followed by a port number. (So therefore you know what the **RESOURCE DATA LENGTH** is.) A successful response to the query should be sent to this location.

We highlight several points of possible confusion:

- These peer-to-peer query messages are sent from your DNS servers to other DNS servers. They are never sent or received by your client from Part A or by DNS clients like `nslookup` and `dig`.

- The P2PQUERY format is used only for Queries and should never appear in Responses. Once the P2P network finds a matching name, it sends a standard Response message as in Part A (see below).

- We need to take the nonstandard step of including resource records in the **ANSWER SECTION** in a Query because the DNS protocol’s **QUESTION SECTION** doesn’t have room for all the information we need for peer-to-peer queries. For the purposes of this project, abusing the DNS protocol format in this way is more convenient than going to the trouble of defining and parsing an entirely new message format.

**Server operations**

Your server should operate as follows:

- When it receives a standard DNS query for a domain name it doesn’t know, it sends to every neighbor a peer-to-peer query message as described in the previous section. It sets the **packet limit** in these messages to `INIT_PKT_LIMIT`, and sets the **reply-to address** to the IP address and port of the client (such as `nslookup`) which originally queried your server.
When your server receives a DNS Query message with a P2PQUERY resource record, it checks for the name in its local database.

- If it has the name, it sends a standard DNS Response message back to the reply-to address.
- Otherwise, it checks the packet limit \( p \) in the query. If \( p = 0 \), it simply does nothing. If \( p > 0 \), it is allowed to send \( p \) packets on this request’s behalf. It recursively sends the P2PQUERY message along to its neighbors, other than the server from which it received this P2PQUERY. Before resending the message, it should appropriately reduce the packet limit: if this server sends to \( n \) neighbors, then it should set the packet limit to \( \frac{p-n}{n} \) in each outgoing message. (There are some corner cases: \( \frac{p-n}{n} \) may not be an integer, and it might happen that \( 0 < p < n \). What do you think is a good way of handling these cases?)

Use the following value for the constant mentioned above:

- \( \text{INIT}_-\text{PKT}_-\text{LIM}_\text{IT} = 10 \)

4 Bonus!

In the spec described so far, the result of a query to the peer-to-peer network goes directly back to the requesting client. Suppose we want to implement on-demand caching: when a server gets a query for a name it doesn’t know, it sends it out recursively using the P2P method but caches the result. Next time it gets the same query, it will know the result immediately assuming the TTL has not expired.

How would you accomplish this? You should be able to do it using only the message and Resource Record formats that we introduced in Part A or this spec, but using them in a slightly different way than was described above. In particular, your solution should be interoperable with servers that don’t implement on-demand caching.

If you choose to implement on-demand caching, describe your approach briefly in the README file accompanying your submission. In particular, describe what you needed to change in the server.

5 Submission and grading

Your submission should be a directory containing:

- The source code of your project (no executables).
- A Makefile, so that running make compiles the server executable (server) and a client executable (client). The client is unchanged from Part A.
- A brief README explaining the design of your server and client.

Submit your project on the instructional machines using the command submit proj2B.

The submitted project needs to compile and run on the UNIX instructional machines. Both the client and server should be written in C or C++.

Grading will be based on passing our test cases, plus 10% extra for test cases testing the bonus.

6 FAQ

What happens if you have the same name installed at multiple servers? One of them will be lucky enough to reply first, and its name → IP mapping will be the one used by the client. We haven’t included any sort of distributed consistency or permissions in this project spec.
Are the integers big or little endian? There exists a standard network byte order which is big endian. Unless otherwise specified, all integers, including port numbers and TTLs, are in network byte order. You can use standard C library functions htons, ntohs, htonl, and ntohl to convert between the local host’s byte order and network byte order, so that your code never needs to know what the network byte order actually is.

How do we insert entries in the server? Your server in Part B has all the functionality of a Part A server. So you should use your client from Part A to install the name → IP mappings.

Should P2PQuery be sent to the tracker or not? Our current tracker implementation will not respond to P2PQuery. Thus it is better if you don’t send it P2PQuery. However, whether you sent P2PQuery to the tracker will not affect your grade.

Should keepalives be sent to the tracker or not? Yes, otherwise the tracker would have no way of knowing when your server is online. In general you can treat the tracker just as any other neighbor, except that it never expires from your neighbor table and you don’t need to send it P2PQuery.

What are the parameter bits for Peer and P2PQuery messages? The parameter bits follow the specification from Part A of the project. More explicitly:

- Bit 0 is set to 0 in queries, and 1 in responses. As noted above, Peer keepalive messages are responses and Peer messages sent to the tracker requesting another neighbor are queries. P2PQuery messages are always queries; the response is a standard DNS response (as in Part A of the project) sent back to the client.
- Bit 5 signifies that the answer is authoritative. Since this has no meaning for the Peer and P2PQuery messages, it should be ignored in Peer and P2PQuery messages you receive, and should be set to 0 in Peer and P2PQuery messages that you send.
- All other bits in the parameter field should be ignored in messages you receive, and should be set to 0 in messages that you send.

Can a keepalive have an entry in the Question section? In keepalive messages that you send, you don’t need to put anything in the Question section. In keepalive messages that you receive (i.e., a message with an entry of type Peer in the Answer section), you can ignore anything that may appear in the Question section.

Should we contact the tracker immediately on startup? Yes, you should send the tracker a keepalive and ask for more neighbors immediately when your server starts up, and periodically thereafter as described above.

The tracker tells my server about itself or one of its existing neighbors. Right. When the tracker gets a Peer query, it sends back a Peer response with a random one of its own neighbors. You should not add the server to your neighbor table if the response is either your own server, or an existing neighbor of your server (i.e., no duplicate entries).

Should the server request all of its needed neighbors at one time? No, only one each KEEPALIVE_INTERVAL. So it may take several rounds before your server acquires enough neighbors.

What response does the server send to the client if it doesn’t know the name? If your server doesn’t know the name locally, it sends out P2PQuery messages. Hopefully at least one other server will know the name and send the answer back to the client. If not, the client gets no reply and will time out.

Isn’t it possible for loops to occur in the routing of P2PQuery messages? Yes. The packet limit will deal with this by limiting the total number of times that the query can be forwarded. So it may loop, but not infinitely. Relatedly, remember you never forward a P2PQuery message back to the server who sent it to you.

How should our server find its own IP address? See the code posted to the Announcements page.
When we cache DNS mappings for the Bonus, does it have to remember what is authoritative? No, we don’t care what you set the authoritative bit to.

Can we assume `select()` updates the timeout to be the remaining time? No. Assume that the timeout is undefined after `select` returns; i.e., `select` might leave it alone or mangle it and you don’t know which it will do.

Must we ensure the tracker reliably receives our keepalives? You don’t need to retransmit keepalives in response to losses; in fact, there’s no good way for your server to tell when a single keepalive has been lost. With keepalive messages, reliability is achieved through repetition every `KEEPALIVE_INTERVAL`. But you do need to pick a good TTL in those keepalive messages, so that your entry in your neighbors’ neighbor tables doesn’t expire just due to the loss (or delay) of one keepalive.

Will there be a tester for this project? Yes, we released one. See the Announcements page.

Are we responsible for picking “good” values for the TTL and packet limit in the case $0 < p < n$? Yes (but this is not supposed to be particularly difficult, so don’t overthink it).

Is the packet limit really the maximum number of packets, or is it an approximation? It’s not an approximation. It is the maximum number of packets that should be sent on behalf of that request.

For the bonus, should we cache only direct client queries or also P2PQueries? You can cache them all if you want, but the only requirement is that when your server gets a request directly via `nslookup` or some other client (rather than from a P2PQuery), then it caches the results (which it gets via P2PQueries).

Do we really send `INIT_PKT_LIMIT` packets to all neighbors or do we divide it up? When initiating a P2PQuery you send $p = \text{INIT_PKT_LIMIT}$ to all neighbors. It probably would have made more sense to design the spec differently, dividing `INIT_PKT_LIMIT` among all neighbors. But, you should stick to what the spec says.

`nslookup` gives an error when it gets responses to P2P queries. That’s expected behavior: it is confused because it directed the query to server $A$ and got a response from some other server $B$. With our testing package (see Announcements page) we included a client which does not have this problem, which you can run as follows:

```bash
./client server_ip server_port name_to_query
```

Also, note that if you implement the bonus, you won’t have this problem.

When finding our IP address, can we assume it’s the second one? When working from the sample code that we gave you, you should filter out the loopback (127.0.0.1) and use the other IP address. You may assume that there is only one non-loopback address on the list. It’s not guaranteed to be the second one all the time, though.

Can we use domain names in keepalive messages we receive as unique IDs? Yes, they are supposed to be unique per server instance, so you could use them internally in your server as unique identifiers for your neighbors. However, a better thing would probably be to use the pair of IP address and port.

What TTL do we pick in keepalive messages? The keepalive TTL is left to you to pick—not an arbitrary value, but a good value. The key phrase from the project spec is: “What do you think the TTL should be set to, in order to maintain the network topology reliably as well as not keep around too many stale entries?” In other words the TTL value should be such that (1) your server’s entry in its neighbors’ neighbor tables doesn’t expire just due to the loss or delay of one keepalive, which can happen because UDP messages are unreliable; and (2) if your server does leave/fail/exit, then within a reasonable amount of time its entry will be cleared out of its neighbors’ neighbor tables.

When $p < n$, can we send fewer than $p$ messages? The best thing to do in this case would be to send one message to each of $p$ neighbors where $p$ is the remaining packet limit, so that you entirely use the packet limit that you are given.
Should our server be robust to malicious activity? You should be in the habit of always writing your code so your server will not crash, regardless of what bits it receives in incoming messages! But, unlike Part A, we will not send malformed messages in our testing. Our tests will focus on correct functionality.

When submitting, do we include the client from Part A? No, you don’t need to include your name-installing client.

There are two domain names in P2PQuery messages. What if they don’t match? Yes, there are two domain names, one in the question section, and then repeated in the answer section along with more info. They both are the name being queried and they should be identical. If they are

Our target is $\geq 4$ neighbors—including the tracker, or not? Four neighbors including the tracker (but our tests won’t penalize you if you used 4 neighbors excluding the tracker). In general you can treat the tracker just as any other neighbor, except that it never expires from your neighbor table and you don’t need to send it P2PQUERIES.