Datanet, Spring 2011, DIKU
3rd Assignment - Anonymizing Distributed Proxy

Truls Asheim
truls@diku.dk

ABSTRACT

This report describes the implementation of and various considerations regarding a HTTP proxy server which has the ability to operate as a part of an anonymizing distributed network which routes each request through a number of nodes masking its origin. Both the server itself and various problems related to the protocol used by the network will be covered.

1. DESIGN

The design of this server is organized as an Erlang OTP supervision tree where each major component of the server is monitored by supervisors which takes care of restarting the servers if anything goes wrong. This is consistent with the Erlang philosophy of failing early and starting over as opposed to attempting to do possibly incomplete error handling.

The major components of the server are:

prtracker This server is responsible maintaining the domain whitelist, the nodelist, and the node blacklist. It also registers the proxy against the tracker periodically.

prnodeparser contains the code for parsing the pure-text version of the peerlist. It receives request from `pr_tracker` whenver a new version of the trackerlist needs parsing.

prsup is a supervisor which is responsible for a number of `pr_servers` (see below). It spawns a new `pr_server` for every request received.

prrootsup This is the supervisor of all the previously mentioned components. It makes sure that they are running at all times and is responsible for restarting them if they crash.

prserver This is the main component of the server which is responsible for processing

The packaging of the server is quit messy. This will hopefully be fixed in the next assignment. See the file `README` in the folder `datanetproxy-0.1` for information on how to compile and run the code.

2. SETUP

I am running my proxy on my own server hosted by Hetzner online in Germany. It is connected to a to the outside world through a 100mbit uplink. It has ports open to the outside world.

3. PROTOCOL LIMITATIONS

The current tracker protocol has two (in my opinion) major flaws:

1. The tracker keeps peers listed for way to long. This causes peers which are no longer active to remain in the list for a long time requiring each peer to keep extensive blacklists containing peers which for various reasons aren't responding correctly. The tracker should expire peers shortly after the minimum registration interval.

2. The fact that the tracker locks you out if you register more frequently than the min-wait interval can be problematic in many circumstances.

Furthermore, the general implementation quality of the peers in the network seems to be quite low causing a high fail-rate for requests and slow transfers. One way of combating this could be to implement a distributed blacklist where the peers could use the registration process to provide the tracker with a list of newly discovered bad peers.

I have also observed a few peers which persistently produces erroneous responses which on the surface seems correct. For instance, I observed one peer which responded to every request with a HTTP 500 (Internal Server Error) status code. The problem with these kinds of responses is that they, on the surface, looks correct. Continuing our previous example it is hard to differentiate a HTTP 500 response produced by a rouge proxy and a "legitimate" HTTP 500 error produced by the destination host. As previously suggested one way of handling this problem could be to implement a distributed

One major annoyance when working with the protocol was that the tracker holds on to nodes for way to long

4. MISBEHAVING PEERS

As noted in the previous section, misbehaving peers is a major problem in the network. In order to have any hope of providing a reasonable performance each peer must maintain it's own blacklist. I use the following method in order to maintain a blacklist of peers which should be avoided:

- Whenever a peer causes a connection timeout (see below) it is added to the blacklist and removed from the list of working peers and added to the blacklist

- Whenever the `pr_tracker` components updates the list of trackers it makes sure that no blacklisted peer reenters the list of peers *unless* the peer has registered with the tracker since it was blacklisted.

- After a new peerlist has been downloaded from the tracker it is checked against the blacklist to ensure that peers which have been removed by the tracker also gets removed from the blacklist.

There are two kinds of timeouts; connection timeouts and response timeouts. Peers which causes timeouts by refusing connections should obviously be blacklisted. The peers which produces response timeouts are more of a gray area which is primarily divided into two groups:

1. Peers which accepts a connection and then never responds due to an error in peer and

2. Peers which are in fact functioning but gets unlucky and chooses a non-reponding peer as it's next hop.

The first group should be blacklisted, but doing so with the second group would be unfair. In my implementation I have therefore decided to not blacklist any of these peers and instead send back a 504 "Gateway timeout" error back to the client. One way of easing differentiation between these two groups would be to implement mandatory timeout values across the network which shouldn't be exceeded by any request.

5. SECURITY MEASUREMENTS

The proxy server is only willing to process requests for domains listed in the whitelist as provided by the tracker. This is ensured by performing a check on every request domain making sure that it is listed in the whitelist. If the proxy receives a request for a domain which is not in the whitelist it will respond with a 304 "Forbidden" error and a custom error page.

6. TESTING

All tests has been performed with the `Max-Forward` header set to 3. The proxy server automatically enforces this for all requests lacking the header.

6.1 General browsing

These times have been found using Firebug with the extension YSlow which shows the total load-time of a webpage. My proxy is currently unable to handle pages with gzip encoding and therefore not all pages in the whitelist are browsable. All the tests below has been performed on the main pages of the respective domains.

Domain	Times tested	Average
www.slashdot.org	3	100.318s
www.reddit.com	3	114s
www.wikipedia.org	3	59.717s
www.vimeo.com	3	77.07s
www.xkcd.com	3	66.155s

6.2 Bandwidth

The bandwidth tests has been performed using `curl(1)`. The reported average is the average of the average transfer speeds reported by `curl(1)`. The following command was used for the benchmark:

```
http_proxy=213.239.205.47:8000 curl \
http://upload.wikimedia.org/wikipedia/\
commons/5/52/Edit_01-12-09_small.ogg > /tmp/foo
```

The test results for the bandwidth test are included below

Run no.	Time	Avg speed
1	226s	137k
2	499s	63.8k
3	106s	291k
4	62s	495k
Averages	225.25s	246.7