Connectbyname and the Proxy Control option

Philip Homburg <philip@nlnetlabs.nl>
Connectbyname (1)

Given a host name and service, return a socket:

\[ s = \text{connectbyname}(\text{hostname}, \text{service}); \]

A bit more complex, with configuration:

\[
\text{cbn\_init}(&\text{cbn\_ctx}); \\
r = \text{connectbyname}(&\text{cbn\_ctx}, \text{hostname}, "http", &s);
\]
Now event-based:

event_base= event_base_new();
cbn_init(&cbn_ctx, event_base);
r= connectbyname_asyn(&cbn_ctx, hostname, "http",
    callback, &state, &ref);
r= event_base_dispatch(event_base);
With DNS policy, and bufferevent:

```c
static void callback(struct bufferevent *bev, void *ref);
static void error_cb(struct cbn_error *error, void *ref);

event_base= event_base_new();
resolver1.settings=
    CBN_AUTHENTICATED_ENCRYPTION |
    CBN_DEFAULT_DISALLOW_OTHER_TRANSPORTS |
    CBN_ALLOW_DO53 | CBN_ALLOW_DOT; | CBN_ALLOW_DOH2; |
resolver1.domain_name= "dns.example.org";
resolver1.svcparams= "no-default-alpn alpn=h2 mandatory=no-default-alpn,alpn";
resolver1.interface= NULL;

cbn_policy_init2(&policy, "name", 0);
cbn_policy_add_resolver(&policy, &resolver1);
cbn_init2(&state.cbn_ctx, &policy, "name", 0, event_base);

r= connectbyname_asyn(&state.cbn_ctx, hostname, "https",
    callback, error_cb, &state, &ref);
r= event_base_dispatch(event_base);
```
Background

- NLnet Labs worked on a connectbyname implementation under a grant from the NLnet Foundation. At the moment just a prototype.
- Asynchronous, Happy Eyeballs, DANE
- On top of getdns
- https://nlnetlabs.nl/projects/connectbyname/about/
Modern Stub Resolver

+------------------+
<p>| |
|                  |
| Application      | UDP/TCP port 53 |</p>
<table>
<thead>
<tr>
<th></th>
<th>DNS over TLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>DNS over HTTP/2</td>
</tr>
<tr>
<td></td>
<td>DNS over HTTP/3</td>
</tr>
<tr>
<td>Stub</td>
<td>DNS over QUIC</td>
</tr>
<tr>
<td>Resolver</td>
<td>Oblivious DoH</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Recursive</td>
<td>Resolver</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
+------------------+
Problems

▶ Many applications use a stub resolver. How many libraries will implement all transports?
▶ DNS directly over UDP has almost no state. DoT, DoH, and DoQ require connection set up.
▶ Load on recursive resolver.
▶ Bad for shortlived applications, for example ping.
Solution

A proxy on the same system as the application.
Examples of existing proxies: unbound, stubby, dnsdist, dnsmasq, systemd-resolved.
Applications have no control.

- How to specify that encryption is required?
- How to specify a DoH connection to a specific public DNS resolver?
- Feedback to the user if DNS resolution fails.
- Diagnostic tools
Proxy Control Option

New EDNS(0) Option
https://datatracker.ietf.org/doc/draft-homburg-dnsop-codcp/

- Stateless, send proxy control options in every request
- Maximize control of the application
- Potential for caching
- Potential for local policies in the proxy
- No DNSSEC validation requirements for the proxy
- Proof of concept:
  https://github.com/getdnsapi/getdns/tree/philip-proxy-config
let mut msg = MessageBuilder::from_target(
    StaticCompressor::new(
        StreamTarget::new_vec()
    ).unwrap();
).unwrap();
let mut msg = msg.question();
msg.push((&self.args.qname, self.args.qtype)).unwrap();
let mut message = msg.as_builder_mut().clone();

let mut tcp = TcpConnection::connect(self.upstream).await.unwrap();
let work_fut = tcp.worker();
let query_fut = tcp.query(&mut message);

tokio::select! {
    work = work_fut => {
        panic!("worker completed");
    }
    reply = query_fut => {
        println!("got reply");
        self.print_response2(reply.unwrap());
    }
}
Feedback, Questions?

Contact me at <philip@nlnetlabs.nl>