CNAME-based Redirection Design Notes

When we configure a redirect type of local-zone or access-control action, we might want to specify a CNAME as the action data, whose canonical name is managed by an external organization. For example, we might configure the following local-zone:

```
local-zone: example.org redirect
local-data: "example.org. CNAME redir.example.com."
```

expecting a query for a.example.org/A will be answered with the CNAME and an A RRset of redir.example.com (where we assume example.com is maintained by an external organization, not the administrator of this unbound instance). The local administrator could resolve redir.example.com and configure unbound with the final answer, but to do so any changes in the maintainer of example.com will have to be applied to the unbound configuration. It would be much more operationally convenient if unbound dynamically resolved the final answer and included in the answer to the local-zone query.

The current unbound implementation accepts the above syntax (also for `access-control-tag-data`) but it doesn't work as described above: it only returns the CNAME for type-CNAME queries; it does not match other types of queries or dynamically resolve the rest of the CNAME chain. This note is a proposal to make it possible.

Higher-Level Design

The basic idea of this proposal is to keep the CNAME from local-zone or access-control-tag-data in the context of a query if the query matches the CNAME answer instead of completing the answer at that point. The local-zone implementation will also need a small change to allow the general-type CNAME matching (not only for the exact-type match).

If the local-zone processing is finished with the CNAME information for a query, it then tries to complete the rest of the CNAME chain either from the cache or by recursive resolution. Once the resolution for the rest of the chain is completed, the answer to the original query will be constructed with the local-zone or access-control CNAME followed by the resolved chain. Note that the local CNAME is neither used for the recursive resolution nor cached. For the iterator or cache point of view, this is no different from a query for the CNAME target name (in the above example, a query for `redir.example.com/A`). This way we can avoid “cache contamination” with local data. Note also that normal clients can always ask `redir.example.com/A` independently (in fact, if the client is a downstream “forwarder” and we return the CNAME without the rest of the chain, the client will send this query to complete the rest of the chain), so there shouldn't be any “contamination” concern in resolving or caching this query simply because it's triggered internally.

We'll also update the configuration validation code to reject configurations that would result in "multiple CNAMEs" or "CNAME and other data", which is prohibited in RFC2181. The revised validation code rejects `local-data` for a redirect local zone if the added data is of CNAME and there's already `local-data` for the zone, or if there's already a CNAME `local-data` for the zone (regardless of the type of the added data). It also applies the same restriction to `access-control-tag-data` for a redirect tag action. Note that this restriction may not be enough since the owner name of the CNAME may be the "origin" of a local zone (this is always the case for `local-data`) in that the origin name is assumed to have some mandatory types of records (at least SOA, and in practice, also NS) so a CNAME can never exist there. However, since the whole concept of `local-zone`s is generally deviant in terms of standard compliance, we chose to allow this additional "violation" for the convenient of users. Note also that due to the restriction described here an SOA local-data cannot be defined for a redirect `local-zone` if CNAME is defined, or vice versa.

Lower-Level Design

There can be several specific ways to implement the high-level design idea. Our current implementation adopts the following approach.

Extend the `query_info` structure

We add a new field to the structure to maintain the local CNAME-related information:

```
struct query_info {
    …
    struct local_rrset* local_alias; /* RRset for the local CNAME */
};
```

This new field is basically just a placeholder until it's finally used to encode the response message (see below). It shouldn't affect cache lookup or recursive resolution; so it's intentionally excluded from the compared values in `msgreply.c:query_info_compare()` .

We use `local_rrset` instead of `ub_packed_rrset_key` as the type of `local_alias` in case we want to eventually support a chain of local aliases (see the limitation section below), in which case the chain will be built through the `next` link. Other than for this reason this could simply be of `ub_packed_rrset_key` .

Update local-zone handling implementation

The `find_tag_datas()` and `local_data_answer()` functions need to be updated so that they match any type of queries with a CNAME local data. In this case, the passed `query_info` will be updated so its alias member points to the CNAME RRset. Also, `local_data_answer()` will skip encoding the data when non-exact CNAME matching happens.

Update high-level query handling implementation

The worker_handle_request() function is updated so that on return from local-zone processing if the query finds an alias (CNAME) it replaces `qname` with the RDATA of the (head of) loca_rrset, which is the CNAME's target name. From this point, this query is handled as if it were a query for the CNAME target name until the final answer is constructed. If the answer is not found in the cache and recursive resolution is necessary, the `mesh_state_create()` function makes a new copy of `local_alias` member data from its regional allocator.

Update answer construction

Some of the functions in `msgencode.c` will have to be updated. First, `insert_query()` uses the owner name of `query_info`'s `local_rrset` if it's non-NULL as the actual query name instead of `qname` to set up the question section and compression tree. The `reply_info_encode()` function now checks if the passed `query_info` has a non-NULL `local_alias` member, and if so, first adds the CNAME RRset stored in the alias to the answer section before the rest of the answers stored in the passed `reply_info.` This change should basically work both for the answer from the cache or as a result of recursive resolution. However, `mesh_send_r eply()` also has to be updated so we don't accidentally reuse encoded data buffer for consecutive two queries whose `local_alias` is different.

Current Limitation

The current implementation doesn't chase CNAME chains within local-zones. For example, `redir.example.com` could find an answer in the local-zone, but this implementation doesn't handle such a case. It wouldn't be very difficult to support such cases, but in the initial implementation I chose to keep the implementation simpler. It might also be operationally sufficient in practice; the local operator can configure a "compressed" CNAME if all related data are in the local configuration. Unlike the case of data maintained in an external organization, there is no concern on how to keep up with changes to the CNAME target.

Expected Behavior

Assume the following unbound configuration:

```
local-zone: example.com static
local-data: "example.com. CNAME cname.example.org."
```

In what follows, we generally (i.e., unless noted otherwise) assume cname.example.org. is not configured as (part of) a local zone and requires external resolution. We also generally assume example queries are for type A. For brevity, we omit unimportant query parameters such as the server address.

- If the CNAME target exists, the response should contain both the CNAME and its target. It should have the AA bit on.

```
% dig example.com
flags: qr aa rd ra
example.com. CNAME cname.example.org.
cname.example.org. A 192.0.2.1
```

- If the CNAME target name exists but queried type does not, it should contain the CNAME only. AA bit on.

```
% dig example.com aaaa
flags: qr aa rd ra
example.com. CNAME cname.example.org.
```

- If the CNAME target name does not exist, the response should indicate NXDOMAIN. The answer section contains the CNAME RR only. AA bit on.

```
% dig example.com
RCODE=NXDOMAIN
flags: qr aa rd ra
example.com. CNAME cname.example.org.
```

- If the attempt of resolving CNAME target results in SERVFAIL, the original query should also result in SERVFAIL. It shouldn't contain any record in the answer section. The question section should contain the original query (it should not replace the qname with the CNAME target). AA bit OFF.

```
% dig example.com
flags: qr rd ra
RCODE=SERVFAIL
;; QUESTION SECTION
; example.com. IN A
```

- If the target record is in a DNSSEC-signed zone, unbound is configured so it can validate the signatures, and the original query enables DNSSEC, then the response should include RRSIGs for the target. AA bit on, but AD bit OFF.

```
% dig example.com +dnssec
flags: qr aa rd ra ; (but no 'ad')
example.com. CNAME cname.example.org.
cname.example.org. A 192.0.2.1
cname.example.org. RRSIG A ...
```

- All of the above should be the same whether unbound receives the query with empty cache or it already has the CNAME target in the cache. The latter case can be confirmed first send a query for "cname.example.org".
- If unbound starts with empty cache, receives a query with RD bit off, and has cached a referral to resolve the CNAME target, then it returns just CNAME and the deepest referral. An example of such referral is the NS RRset of the root name servers. (Don't try to test that with a forwarder - use a faked root server).

```
% dig . ns
(this makes sure there’s at least one referral in the cache)
```

```
% dig example.com +norec
flags: qr aa ra
;; ANSWER SECTION:
example.com. CNAME cname.example.org.
;; AUTHORITY SECTION:
. IN NS a.root-servers.net.
...
```

- If unbound receives a query for the CNAME target directly while trying to resolve it as a result of CNAME-based redirection, or if it receives a query subject to a CNAME-based redirection while resolving a direct query for the target, then unbound should unify the two resolution attempts (here we assume it only uses 1 worker thread). But the final answers should be different as the query names are different.

```
% dig example.com
(while unbound is trying to resolve cname.example.org)
% dig cname.example.org
(there should be only one external query for cname.example.org from
unbound.  and both attempts of dig should succeed correctly).
```

- All of the above should also be the same for tag-based redirection.