How we are developing a next generation DNS API for applications

Sara Dickinson         Willem Toorop
Sinodun                  NLnet Labs

OARC Spring Workshop 2016
Introduction

- Present an implementation of the getdns API
- Why is a new DNS API needed?
- Goals and evolution
- Key Features (for Applications and for DNS)
  - Practical Examples
- DNS is a moving target – research tool
getdns overview

- getdns is a modern asynchronous DNS API
- Designed by and for application developers
- First specification by Paul Hoffman 2013
- First Open Source implementation developed by a collaborative effort:
  - Verisign Labs, NLnet Labs
  - No Mountain, Sinodun

BIG NEWS: 1.0.0b1 is now available!
getdns overview

- Offers stub and full recursive mode (libunbound)
- All record types and fine-grained access to response
- DNSSEC validation (even in stub mode)
- Supports DNS Privacy (DNS-over-TLS)

- Implemented in C with bindings: Python, nodejs, Java, PHP
- Homepage: https://getdnsapi.net/
- Spec: https://getdnsapi.net/spec.html

More details later...
Why was it needed?

- Default OS DNS resolver libraries (getaddrinfo(), getnameinfo()) are slow to evolve and don’t support modern DNS capabilities.

[Box: DNSSEC/DANE, DNS Privacy, ASYNC]

- **Catch 22**: No nice APIs for applications, no uptake of new features, no drivers for deployment...

- **Light Bulb moment**: API’s were designed by and for DNS folks!

[Image: Great Scott!!]
Goals

• Goal of getdns

… API design from talking to application developers …

… create a natural follow-on to getaddrinfo()

“a particular hope is to inspire application developers towards innovative security solutions in their applications”
Evolution

- Core dev team, that has owned the spec since 2014

- Bindings have evolved with core code and spec

- Code taken to multiple Hackathons/conferences to get direct feedback from application developers
  - TNW (The Next Web), W3C, PyCon, IETF

- Discussions with mobile and embedded platforms to understand needs (minimal dependencies)
Key features

https://getdnsapi.net/query.html

- Overview of features
- Requirements
- Solution
What Application Developers Want

- Async by default. Why?
  - Modern applications organized around events
    - File system and Network IO
    - User interaction
    - Start looking up names in advance
    - Schedule requests in parallel
  - Spin on an event loop
What Application Developers Want

- Async by default in getdns
  - Requests are scheduled
  - No 'execution' (i.e. no 'run event loop')

*from the spec ...

“Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used”

- We provide extensions for libevent, libev, libuv
getdns_address

getdns_return_t getdns_address (getdns_context *context,
                             const char *name,
                             getdns_dict *extensions,
                             void *userarg,
                             getdns_transaction_t *transaction_id,
                             getdns_callback_t callbackfn);

typedef void (*getdns_callback_t)(getdns_context *context,
                                  getdns_callback_type_t callback_type,
                                  getdns_dict *response,
                                  void *userarg,
                                  getdns_transaction_t transaction_id);
function setup_tls(conn, err, res)
{
    conn.socket = tls.connect(443, {host: res.just_address_answers[0], rejectUnauthorized: false, servername: conn.name}, function() { verify_tlsa(conn, null, null)});
}

var conn = { name: 'getdnssapi.net', socket: null, tlsa_rrs: null};

ctx = getdns createContext();

cxt.address( conn.name, function(err, res) { setup_tls(conn, err, res) });

cxt.general( '_443._tcp.' + conn.name, getdns.RRTYPE_TLSA, { dnssec_return_only_secure: true }, function(err, res) { verify_tlsa(conn, err, res) });
What Application Developers Want

• Hand control to the application

• Custom/User defined Event Loops
  • From getdns version 1.0.0beta and upwards linked against libunbound version 1.5.9 and upwards:

  Event loop is also propagated to recursive resolution

• Custom/User defined Memory Management

• See Appendix for details of both

CRUCIAL for Integration
- nodejs
- Windows
What Application Developers Want

• JSON dict like interfaces to DNS data. Why?

• Makes programming easy – you see what's there

OUTPUT: response dictionary

```json
{
    "answer_type": GETDNS_NAMETYPE_DNS,
    "status": GETDNS_RESPSTATUS_GOOD,
    "canonical_name": <bindata of "www.getdnsapi.net.">,
    "just_address_answers":
    [ { "address_data": <bindata for 185.49.141.37>,
      "address_type": <bindata of "IPv4">
    },
    { "address_data": <bindata for 2a04:b900:0:100::37>,
      "address_type": <bindata of "IPv6">
    }]
},
"replies_full":
[
    <bindata of 0x00008180000100020004000103777777...>,
    <bindata of 0x00008180000100020004000903777777...>
],
"replies_tree":
[
    { ... first reply ... },
    { ... second reply ... }
]
```
getdns: JSON dict

- JSON dict like interfaces to DNS data. Why?
  - Makes programming easy – you see what’s there

---

```json

"replies_tree": [  
  {  
    "header": {  
      "qdcount": 1,  
      "ancount": 2,  
      "rd": 1,  
      "ra": 1,  
      "opcode": GETDNS_OPCODE_QUERY,  
      "rcode": GETDNS_RCODE_NOERROR,  
      "...  
    },  
    "question": {  
      "qname": <bindata for www.getdnsapi.net.>,  
      "qtype": GETDNS_RRTYPE_A,  
      "qclass": GETDNS_RRCLASS_IN,  
      "...  
    },  
    "answer": [  
      {  
        "name": <bindata for www.getdnsapi.net.>,  
        "type": GETDNS_RRTYPE_A,  
        "class": GETDNS_RRCLASS_IN,  
        "rdata": {  
          "ipv4_address": <bindata for 185.49.141.37>,  
          "rdata_raw": <bindata of 0xb9318d25>  
        },  
        "...  
      },  
    ],  
    "authority": [...],  
    "additional": [],  
    "canonical_name": <bindata of "www.getdnsapi.net.">,  
    "answer_type": GETDNS_NAMETYPE_DNS  
  },  
  {  
    "header": {  
      "...  
    }  
  }  
]

OUTPUT: response dictionary – replies trees
```
getdns: JSON dict

- JSON dict like interfaces to DNS data. **Why?**
- Extensible (allows experimentation)

```json
{
    "dnssec_return_validation_chain": GETDNS_EXTENSION_TRUE,
    "specify_class": GETDNS_CLASS_HS,
    "add_opt_parameters": {
        "maximum_udp_payload_size": 1232,
        "options": [
            {
                "option_code": 10,
                "option_data": cookie_bindata
            }
        ]
    }
}
```
getdns: JSON dict

- JSON dict like interfaces to DNS data.
- *(almost)* all data is in wire format
- The bindata's just point to the right spot in the packet *(JIT potential)*

```
"replies_tree": [
  { "header": { "qdcount": 1, "ancount": 2, "rd": 1, "ra": 1,
    "opcode": GETDNS_OPCODE_QUERY,
    "rcode": GETDNS_RCODE_NOERROR, ...
  },
  "question": { "qname": <bindata for www.getdnsapi.net.>,
    "qtype": GETDNS_RRTYPE_A,
    "qclass": GETDNS_RRCLASS_IN,
  },
  "answer": [ { "name": <bindata for www.getdnsapi.net.>,
    "type": GETDNS_RRTYPE_A,
    "class": GETDNS_RRCLASS_IN,
    "rdata": { "ipv4_address": <bindata for 185.49.141.37>,
      "rdata_raw": <bindata of 0xb9318d25> },
  }, ...
  "authority": [ ... ],
  "additional": [],
  "canonical_name": <bindata of "www.getdnsapi.net.">,
  "answer_type": GETDNS_NAMETYPE_DNS
 },
{ "header": { ...
```
What do C Developers Want

- C-friendly access to JSON dict data
  - Unconventional, too generic, no type safety
  - Lengthy and repetitive to get to the data in C

```c
if ((r = getdns_address_sync(ctx, "getdnsapi.net", ext, &resp)))
    return r;
elif ((r = getdns_dict_get_list(resp, "just_address_answers", &jaa)))
    return r;
elif ((r = getdns_list_get_dict(jaa, 0, &addr_dict)))
    return r;
elif ((r = getdns_list_get_bindata(addr_dict, "address_data", &addr)))
    return r;
```
What do C Developers Want

- C-friendly access to JSON dict data
  - Unconventional, too generic, no type safety
  - Lengthy and repetitive to get to the data in C

```c
if ((r = getdns_address_sync(ctx, "getdnsapi.net", ext, &resp)))
    return r;
else if ((r = getdns_dict_get_list(resp, "just_address_answers", &jaa)))
    return r;
else if ((r = getdns_list_get_dict(jaa, 0, &addr_dict)))
    return r;
else if ((r = getdns_list_get_bindata(addr_dict, "address_data", &addr)))
    return r;
```

```python
resp = ctx.address('getdnsapi.net')
addr = resp.just_address_answers[0]['address_data']
```
What do C Developers Want

• Since getdns 0.5: **JSON pointer access**

• Re-wrote examples in the spec – now only 2 lines in C!

```c
if ((r = getdns_address_sync(ctx, "getdnsapi.net", ext, &resp)))
    return r;
else if ((r = getdns_dict_get_bindata(resp, "just_address_answers/0/address_data", &addr)))
    return r;
```
What DNS Applications Want

- **DANE**
  - Need DNSSEC to get TLSA, SSHFP, OPENPGPKEY etc.
  - **Feedback** Validating stub added early
- Native DNSSEC validation

Uses existing DNS caching infrastructure, but not without problems!
DANE in getdns

• Since getdns 0.5.1: **Roadblock avoidance**
  • Maximise stub usage when possible
  • Fall-back to full recursion when necessary
    • *on a per query basis*
What DNS Applications will Want

- **DANE**
  - Need DNSSEC to get TLSA, SSHFP, OPENPGPKEY etc.

- What else is needed? *(i.e. still hampering DANE deployment)*
  - Verification coming in OpenSSL 1.1.0

- Future work for this API/library

Follow redirects to a service *(CNAME, MX, SRV)*
Collect TLSAs with the reference identifiers *(RFC7671, RFC7672, RFC7673 & RFC6125)*
What DNS Researchers Want

• DNSSEC API that offers validation functions
  • Take control of validation

• Ability to experiment
  • e.g. Custom code new EDNS0 options

• Flexible access to responses
  • Work in progress: DNSSEC transparency
    • draft-shore-tls-dnssec-chain-extension
What getdns offers

- Unique DNSSEC API
  - `dnssec_return_validation_chain` extension
  - `getdns_validate_dnssec()` function

- Possible to use getdns to do EDNS0 cookies before implemented

- Conversion functions:
  - getdns 0.9.0: resource record
  - getdns 1.0.0b1: complete DNS messages

Wire format <-> getdns_dict <-> presentation format
DNSSEC validation in getdns

```c
getdns_return_t
getdns_validate_dnssec(
    getdns_list *to_validate,
    getdns_list *bundle_of_support_records,
    getdns_list *trust_anchor_records
);
```

```json
{
    "validation_chain":
    [ {
        "name": <bindata for .>,
        "type": GETDNS_RRTYPE_DNSKEY,
        ... }],
    {
        "name": <bindata for .>,
        "type": GETDNS_RRTYPE_DNSKEY,
        ... },

    {
        "name": <bindata for .>,
        "type": GETDNS_RRTYPE_RRSIG,
        "rdata": {
            "signers_name": <bindata for .>,
            "type_covered": GETDNS_RRTYPE_DNSKEY,
            ... },
        ... },

    {
        "name": <bindata for net.>,
        "type": GETDNS_RRTYPE_DS,
        ... },
    {
        "name": <bindata for net.>,
        "type": GETDNS_RRTYPE_RRSIG,
        "rdata": {
            "signers_name": <bindata for .>,
            "type_covered": GETDNS_RRTYPE_DS,
            ... },
        ... }
}
```
What meta-data Wants

• Privacy
  • Pervasive monitoring (of DNS) is an attack (RFC 7258, RFC 7626).
  • “Specification for DNS over TLS” is now approved as RFC!

• getdns
  • Transport handling extended from original spec because new specs/standards have emerged (RFC 7766)
  • Transport list with fallback (TLS, TCP, UPD)
  • TLS authentication possible (2 mechanisms)

• I-D: Padding of DNS message
• I-D: EDNS Client subnet privacy
Conclusions

- Spec development – lessons learned
  - Practical input from users
  - Iterative..
  - Must be prepared to learn and adapt

- Hope is *getdns* will change the way DNS lookups are done by applications
  - Async
  - Increased take up of DNSSEC/DANE
  - Increased use of TCP/TLS
The future

- Small cache for the stub (good for DNSSEC, good for roadblock avoidance)
- Sharing upstreams in between contexts good for upstreams that keep connections open
- JIT parsing of response dict – some optimisation
- Put the intelligence for doing TLSA lookups (RFC7671, RFC7672 & RFC7673) in getdns
- Custom RR types rdata fields with a DNS extension language
The future

• 1.0 release is also a natural transition point
  • Focus to date has been API and implementation
  • Challenge now is deployment and further evolution

• Desire to involve wider community as move forward
  • Call for interested parties to become involved in future of getdns
  • Call for ideas for integration into OS distros
Thank you!

https://getdnsapi.net
Appendix
Appendix – Custom/User Defined Event Loops

- Available by including `<getdns/getdns_extra.h>`

```c
typedef struct getdns_eventloop_vmt getdns_eventloop_vmt;
typedef struct getdns_eventloop {
    getdns_eventloop_vmt *vmt;
    /* object data here */
} getdns_eventloop;

getdns_return_t getdns_context_set_eventloop(
    getdns_context* context, getdns_eventloop *eventloop);

/* Virtual Method Table */
struct getdns_eventloop_vmt {
    void (*cleanup)(getdns_eventloop *this);
    getdns_return_t (*schedule)(getdns_eventloop *this,
        int fd, uint64_t timeout, getdns_eventloop_event *ev)
    getdns_return_t (*clear)(getdns_eventloop *this,
        getdns_eventloop_event *ev)
    void (*run)(getdns_eventloop *this);
    void (*run_once)(getdns_eventloop *this, int blocking);
};
```
Appendix - Custom/User Defined Event Loops

/* event data */
typedef void (*getdns_eventloop_callback)(void *userarg);

typedef struct getdns_eventloop_event {
    void *userarg;
    getdns_eventloop_callback read_cb;
    getdns_eventloop_callback write_cb;
    getdns_eventloop_callback timeout_cb;

    /* Pointer to the underlying event */
    void *ev;
} getdns_eventloop_event;
Appendix – Custom memory functions

getdns_return_t
getdns_context_create_with_extended_memory_functions(
    getdns_context **context,
    int set_from_os,
    void *userarg,
    void *(*malloc) (void *userarg, size_t),
    void *(*realloc)(void *userarg, void *, size_t),
    void (*free) (void *userarg, void *)
);
Appendix – DANE validation in OpenSSL

```c
if (! (ext = getdns_dict_create ()))
   ; /* error */
else if ((r = getdns_dict_set_int( ext
   , "dnssec_return_only_secure"
   , GETDNS_EXTENSION_TRUE )))
   ; /* error */
else if ((r = getdns_general_sync( gctx
   , "_443._tcp.getdnsapi.net"
   , GETDNS_RRTYPE_TLSA, ext, &resp)))
   ; /* error */
else if ((r = getdns_dict_get_int( resp,
   "/replies_tree/0/answer/0/rdata/certificate_usage", &usage))
   ; /* error */
else if ((r = getdns_dict_get_int( resp,
   "/replies_tree/0/answer/0/rdata/selector", &selector))
   ; /* error */
else if ((r = getdns_dict_get_int( resp,
   "/replies_tree/0/answer/0/rdata/matching_type", &mtype))
   ; /* error */
else if ((r = getdns_dict_get_int( resp,
   "/replies_tree/0/answer/0/rdata/certificate_association_data", &ca_data))
   ; /* handle error */
else if (! (sctx = SSL_CTX_new( TLS_client_method ())))
   ; /* error */
else if (SSL_CTX_dane_enable(sctx) <= 0)
   ; /* error */
else if (SSL_new(sctx) == NULL)
   ; /* error */
else if (SSL_dane_enable(ssl, "getdnsapi.net") <= 0)
   ; /* error */
else if (SSL_dane_tlsa_add(ssl, usage, selector, mtype, ca_data->data, ca_data->size))
   /* handle error */
```