A new stub resolver

Willem Toorop
A new stub resolver – vBSDcon 2015

Willem Toorop (NLnet Labs)

API is:

• A **DNS API** specification **by and for application developers**
  (for resolving)
  (for application)

• First implementation by **VERISIGN** LABS and **NLnet Labs**

From Verisign:
Theogene Bucuti, Craig Despeaux, Angelique Finan, Neel Goyal, Scott Hollenbeck, Shumon Huque, Sanjay Mahurpawar, Allison Mankin, Sai Mogali, Prithvi Ranganath, Rushi Shah, Vinay Soni, Bob Steagall, Gowri Visweswaran, Glen Wiley

From NLnet Labs:
Olaf Kolkman, Benno Overeinder, Willem Toorop, Wouter Wijngaards

From Sinodun:
Sara and John Dickinson

From No Mountain Software:
Melinda Shore
getdns API is:

- A **DNS API** specification by and for application developers (for resolving)
- First implementation by Verisign Labs and NLnet Labs (for application)

- OpenBSD & FreeBSD already have unbound in system
- getdns might have a role too
A DNS API specification by and for application developers

First implementation by VERISIGN™ LABS and NLnet Labs

Bootstrap encrypted channel (TLS)
from DNSSEC authenticated keys (DANE)
especially applicable/suitable to system software!

- Lack of user interaction (who do you trust)
- Policy published over sidechannel (DNSSEC)
Why?

Issues with the system stub

• System's stub accessed by application via getaddrinfo() & getnameinfo()
Why?

Issues with the system stub

• System's stub accessed by application via `getaddrinfo()` & `getnameinfo()`
• Translate names ↔ numbers (also DNS)

DNS: Domain Name System

*The phonebook of the Internet*
Why?
Issues with the system stub

- System's stub accessed by application via `getaddrinfo()` & `getnameinfo()`
- Translate names ↔ numbers (also DNS)
- What about something other than numbers (i.e. MX, SPF, SSHFP, TLSA, OPENPGPKEY etc.)

DNS: Domain Name System
Global decentralized distributed database for more than just names and numbers.
Why?

Issues with the system stub

- System's stub accessed by application via `getaddrinfo()` & `getnameinfo()`
- Translate names ↔ numbers (also DNS)
- What about something other than numbers (i.e. MX, SPF, SSHFP, TLSA, OPENPGPKEY etc.)
- `libresolv?` (res_query(), dn_comp() etc.)
Why?

Issues with the system stub

- System's stub accessed by application via `getaddrinfo()` & `getnameinfo()`
- Translate names ↔ numbers (also DNS)
- What about something other than numbers (i.e. MX, SPF, SSHFP, TLSA, OPENPGPKEY etc.)
- `libresolv`? (res_query(), dn_comp() etc.)
- Blocks on I/O (no asynchronous DNS)
Why?
Issues with the system stub

• System's stub accessed by application via getaddrinfo() & getnameinfo()
• Translate names ↔ numbers (also DNS)
• What about something other than numbers (i.e. MX, SPF, SSHFP, TLSA, OPENPGPKEY etc.)
• libresolv? (res_query(), dn_comp() etc.)
• Blocks on I/O (no asynchronous DNS)
• No control over I/O (upstreams, transport, how to fallback/timeout, privacy)
Why?
Issues with the system stub

• DNSSEC!
Why?
Issues with the system stub

- DNSSEC!
- A global distributed database with authenticated data
Why?
Issues with the system stub

- DNSSEC!
- A global distributed database with authenticated data
- Wasn't it about protecting users against domain hijacking?

• DNS: *The phonebook of the Internet*
• Data unauthenticated
• DNSSEC to the rescue
Why?
Issues with the system stub

- DNSSEC!
- A global distributed database with authenticated data
- Wasn't it about protecting users against domain hijacking?

  - DNS: *The phonebook of the Internet*
  - Data unauthenticated
  - DNSSEC to the rescue

- Yes, but it does so by giving (origin) authenticated answers
  - where *origin* means that the authoritative party for a zone authenticates the domain names within that zone
Why?
Issues with the system stub

• DNSSEC!
• A global distributed database with authenticated data
• Wasn't it about protecting users against domain hijacking?

• DNS: The phonebook of the Internet
  • Data unauthenticated
  • DNSSEC to the rescue

• Yes, but it does so by giving (origin) authenticated answers
• **How does this concern the stub?**
  – Authentication is interesting for applications
DNSSEC - for applications
- for TLS

- Transport Layer Security (TLS) uses both asymmetric and symmetric encryption
- A symmetric key is sent encrypted with remote public key

- How is the remote public key authenticated?
• How is the remote public key authenticated?

DNSSEC - for applications
- for TLS
DNSSEC - for applications
- for TLS

- Through Certificate Authorities (CAs), maintained in OS, browser...

- Every CA is authorized to authenticate for any name (as strong as the weakest link)

- There are 650+ CAs
(See https://www.eff.org/observatory)
DNSSEC - for applications
- for TLS

- DNS-based Authentication of Named Entities (DANE) RFC6698
Why?
Issues with the system stub

• DNSSEC!
• A global distributed database with authenticated data
• Wasn't it about protecting users against domain hijacking?
  • DNS: *The phonebook of the Internet*
  • Data insecure/unprotected
  • DNSSEC to the rescue

• Yes, but it does so by giving (origin) authenticated answers
• **How does this concern the stub?**
  – Authentication is interesting for applications
Why?

Issues with the system stub

- DNSSEC!
- A global distributed database with authenticated data
- Wasn't it about protecting users against domain hijacking?

- Yes, but it does so by giving (origin) authenticated answers
- **How does this concern the stub?**
  - Authentication is interesting for applications
  - DNSSEC deployment is not completely finished yet
DNSSEC - the first mile

Could be your phone
Could be the Wi-Fi

- Is the local network resolver trustworthy?
DNSSEC - the first mile

- Is the local network resolver trustworthy?
- Who's to blame?
DNSSEC - the first mile

- Is the local network resolver trustworthy?
- Who's to blame?

Willem Toorop (NLnet Labs)
DNSSEC - the first mile

- Is the local network resolver trustworthy?
- Who's to blame?
- Application does not know an answer is secure (AD bit not given with getaddrinfo())
DNSSEC - the first mile

- Is the local network resolver trustworthy?
- Who's to blame?
- Application does not know an answer is secure
- Network resolver does not need to validate
DNSSEC - the first mile

- Is the local network resolver trustworthy?
- Who's to blame?
- Application does not know an answer is secure
- Network resolver does not need to validate
- And when it is not even DNSSEC-aware
DNSSEC - the first mile

- Is the local network resolver trustworthy?
- Who's to blame?
- Application does not know an answer is secure
- Network resolver does not need to validate
- And when it is not even DNSSEC-aware

https://www.us-cert.gov/ncas/alerts/TA15-240A

Alert (TA15-240A)
Controlling Outbound DNS Access

Configure enterprise perimeter network devices to block all outbound User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) traffic to destination port 53, except from specific, authorized DNS servers (including both authoritative and caching/forwarding name servers).
Why?
Issues with the system stub

- DNSSEC!


Bootstrap encrypted channel (TLS)

from DNSSEC authenticated keys (DANE)

especially applicable/suitable to system software!

• Lack of user interaction (who do you trust)
• Policy published over sidechannel (DNSSEC)
Why?

Issues with the system stub resolver

- DNSSEC!
- Bootstrap encrypted channel (TLS)
  from DNSSEC authenticated keys (DANE)
  especially applicable/suitable to system software!
- Lack of user interaction (who do you trust)
- Policy published over sidechannel (DNSSEC)

Why?

Issues with the system stub

- DNSSEC!
- (Inband policy assertion susceptible to downgrade attacks)

```
EHLO nlnetlabs.nl
250-getdns.nlnetlabs.nl Hello [IPv6:2a04:b900:0:1:14bc:270e:5c12:6e7b], pleased to meet you
250-ENHANCEDSTATUSCODES
250-STARTTLS
250-PIPELINING
250-8BITMIME
```

Bootstrap encrypted channel (TLS)
from DNSSEC authenticated keys (DANE)
especially applicable/suitable to system software!

- Lack of user interaction (who do you trust)
- Policy published over sidechannel (DNSSEC)
Why?

Issues with the system stub

- [https://github.com/phicoh/openssh-getdns/tree/getdns](https://github.com/phicoh/openssh-getdns/tree/getdns)
- Validates SSHFP with a trust anchor on a default (configurable) location
  (opposed to checking AD bit or using non-standard `resolv.conf` option)
  
  ```
  --with-trust-anchor=KEYFILE
  
  Default location of the trust anchor file.
  [default=SYSCONFDIR/unbound/getdns-root.key]
  ```

- Manage default trust anchor with `unbound-anchor`

*Bootstrap encrypted channel (TLS)*

*from DNSSEC authenticated keys (DANE)*

*especially applicable/suitable to system software!*

- Lack of user interaction (who do you trust)
- Policy published over sidechannel (DNSSEC)
Why?
Motivation by API (spec) designers

- From Design considerations
  
  ... There are other DNS APIs available, but there has been very little uptake ...
  
  ... talking to application developers ...
  
  ... the APIs were developed by and for DNS people, not application developers ...

- Goal

  ... API design from talking to application developers ...
  
  ... create a natural follow-on to getaddrinfo() ...
Why?
Motivation by API (spec) designers

- Goal
  … API design from talking to application developers …
  … create a natural follow-on to getaddrinfo() …

- Current spec: https://getdnsapi.net/spec.html
- Originally edited by Paul Hoffman (published April 2013)
- Mailing-list: https://getdnsapi.net/mailman/listinfo/spec
- Archive: https://getdnsapi.net/pipermail/spec/
- Maintained by the getdnsapi.net team since October 2014
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  - Full recursive via libunbound
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  - Full recursive via libunbound
  - `--enable-stub-only` configure option (no libunbound dependency)
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  - Full recursive via libunbound
  - `--enable-stub-only` configure option (no libunbound dependency)
- Delivers validated DNSSEC even in stub mode (off by default)
  - libldns still (but only) used for
    `ldns_verify_rrsig()` & `ldns_rr_compare_ds_dnskey()`
  - Plan to lift those out before coming major release
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  - Full recursive via libunbound
  - `--enable-stub-only` configure option (no libunbound dependency)
- Delivers validated DNSSEC even in stub mode (off by default)
  - libldns still (but only) used for `ldns_verify_rrsig()` & `ldns_rr_compare_ds_dnskey()`
  - Plan to lift those out before coming major release
- Resolves names and gives fine-grained access to the response with a response dict type:
  - Easy to inspect: `getdns_pretty_print_dict()`
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  ```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 0x0000818000010002000400010377777777...>,
    <bindata of 0x0000818000010002000400090377777777...>
  ],
  "replies_tree":
  [  
    { ... first reply ... },
    { ... second reply ... }
  ]
  ```

- Delivers validated DNSSEC even in stub mode (off by default)
  - libldns still (but only) used for ldns_verify_rrsig() & ldns_rr_compare_ds_dnskey()
  - Plan to lift those out before coming major release

- Resolves names and gives fine-grained access to the response
  - Easy to inspect: getdns_pretty_print_dict()
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  "replies_tree":
  
  
  - "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
  
  - Full recursive via libunbound
  
  - Delivers validated DNSSEC even in stub mode (off by default)
  - libldns still (but only) used for ldns_verify_rrsig() & ldns_rr_compare_ds_dnskey()
    - Plan to lift those out before coming major release

  - Resolves names and gives fine-grained access to the response with a response dict type:
    - Easy to inspect: getdns_pretty_print_dict()
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
  - Full recursive via libunbound
  - `--enable-stub-only` configure option (no libunbound dependency)
- Delivers validated DNSSEC even in stub mode (off by default)
  - libldns still (but only) used for
    - `ldns_verify_rrsig()` & `ldns_rr_compare_ds_dnskey()`
  - Plan to lift those out before coming major release
- Resolves names and gives fine-grained access to the response with a response dict type:
  - Easy to inspect: `getdns_pretty_print_dict()`
    - `getdns_print_json_dict()`
    - `getdns_print_json_list()`
  - Maps well to popular modern scripting languages
Features (& implementation)

- Both stub and full recursive modes (recursive by default)
- Delivers validated DNSSEC even in stub mode (off by default)
- Resolves names and gives fine-grained access to the response
  - Easy to inspect: getdns_pretty_print_dict()
  - Maps well to popular modern scripting languages

Have a look at https://getdnsapi.net/query.html
Features (& implementation)  
DNSSEC extensions

- On a per query basis by setting extensions
- `dnssec_return_status`
  - Returns security assertion. Omits bogus answers
  - `{ # This is the response object
    "replies_tree":
      [
        { # This is the first reply
          "dnssec_status": GETDNS_DNSSEC_INSECURE,
        }
        "dnssec_status" can be GETDNS_DNSSEC_SECURE,
        GETDNS_DNSSEC_INSECURE or
        GETDNS_DNSSEC_INDETERMINATE
      ]
    }

- `void getdns_context_set_return_dnssec_status(context, enable);`
Features (& implementation)

DNSSEC extensions

- `dnssec_return_only_secure` (The DANE extension)
  - Returns security assertion. Omits bogus and insecure answers
  - `{ # This is the response object
      "replies_tree": [],
      "status": GETDNS_RESPSTATUS_NO_SECURE_ANSWERS,
    - Or "status": GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS
Features (& implementation)

DNSSEC extensions

- `dnssec_return_validation_chain`

  ```
  - { # Response object
     "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, ... }, ... }],
  ```

- Can be combined with `dnssec_return_status` and `dnssec_return_only_secure`

- No replies omitted! Only now "dnssec_status" can be `GETDNS_DNSSEC_BOGUS`
Features (& implementation)

- Asynchronous modus operandi is the default
  - From specification section 1.8:
    ... there is no standard method to set the event base in the DNS API: those are all added as extensions ...

    ... Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used.
  - We have provided functions for: libevent, libev, libuv
  - Or without extension: getdns_context_run()
Features (& implementation)

- Asynchronous modus operandi is the default
  - From specification section 1.8:
    ... there is no standard method to set the event base in the DNS API: those are all added as extensions ...

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

    - We have provided functions for: libevent, libev, libuv
    - Or without extension: getdns_context_run()

- Set custom memory management functions
  - For example for regions
  - Beware of heartbleed!
Features (& implementation)

- Asynchronous modus operandi is the default
  - From specification section 1.8:
    ... there is no standard method to set the event base in the DNS API:
    those are all added as extensions ...

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

    - We have provided functions for: libevent, libev, libuv
    - Or without extension: `getdns_context_run()`

- Set custom memory management functions
  - For example for regions
  - Beware of heartbleed!

- Hook your app into getdns
  - Hook into the applications native event base
    ( nodejs bindings & iOS grand central dispatch POC example )
Features (& implementation)

hop-by-hop communication options (for stub)

• add_opt_parameters extension
  – To set arbitrary EDNS0 options
  – Implement DNS cookies with the library
Features (& implementation)
hop-by-hop communication options (for stub)

- add_opt_parameters extension
  - To set arbitrary EDNS0 options
  - Implement DNS cookies with the library

- DNS cookies by the library --enable-draft-edns-cookies
Features (& implementation)
hop-by-hop communication options (for stub)

- **add_opt_parameters extension**
  - To set arbitrary EDNS0 options
  - Implement DNS cookies with the library

- **DNS cookies by the library**
  - `--enable-draft-edns-cookies`

- **TCP Fast Open (RFC 7413)**
  - `--enable-tcp-fastopen`
Features (& implementation)
hop-by-hop communication options (for stub)

- **add_opt_parameters extension**
  - To set arbitrary EDNS0 options
  - Implement DNS cookies *with* the library

- **DNS cookies by the library** `--enable-draft-edns-cookies`

- **TCP Fast Open (RFC 7413)** `--enable-tcp-fastopen`

- **Setting of “tried in turn” transport lists**
  - `GETDNS_TRANSPORT_UDP`
  - `GETDNS_TRANSPORT_TCP`
Features (& implementation)

hop-by-hop communication options (for stub)

- **add_opt_parameters** extension
  - To set arbitrary EDNS0 options
  - Implement DNS cookies *with* the library

- DNS cookies by the library  
  --enable-draft-edns-cookies

- TCP Fast Open (RFC 7413)  
  --enable-tcp-fastopen

- Setting of “tried in turn” transport lists
  - GETDNS_TRANSPORT_UDP
  - GETDNS_TRANSPORT_TCP
  - getdns_context_set_dns_transport_list();

- Special Cookies/TCP/TLS only open resolver for experimentation available on 2a04:b900:0:100::38 and 185.49.141.38
Features (& implementation)
hop-by-hop communication options (for stub)

- nsswitch module! by Theogene H. Bucuti, University of North Texas and Gowri Visweswaran and Allison Mankin, Verisign Labs
Features (& implementation) 
-hop-by-hop communication options (for stub)

- nsswitch module! by Theogene H. Bucuti, University of North Texas and Gowri Visweswaran and Allison Mankin, Verisign Labs
Features (& implementation)

hop-by-hop communication options (for stub)

- nsswitch module! by Theogene H. Bucuti, University of North Texas and Gowri Visweswaran and Allison Mankin, Verisign Labs
Features (& implementation)
hop-by-hop communication options (for stub)

- **nsswitch module** by Theogene H. Bucuti, University of North Texas and Gowri Visweswaran and Allison Mankin, Verisign Labs

Reuse context to reuse statefull transport sessions
# Bindings

- **nodejs** by Neel Goyal  
  (integrated with native async event loop)  
  https://github.com/getdnsapi/getdns-node

- **python** by Melinda Shore  
  https://github.com/getdnsapi/getdns-python-bindings

- **java** by Vinay Soni, Prithvi Ranganath and Sanjay Mahurpawar  
  https://github.com/getdnsapi/getdns-java-bindings

- **php** by Scott Hollenbeck  
  https://github.com/getdnsapi/getdns-php-bindings
from getdns import *

ctx = Context()
ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = ctx.general( '_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res[‘status’] == RESPSTATUS_GOOD:
    # Process TLSA RRs
from getdns import *

cxt = Context()
cxt.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = cxt.general( '_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res[‘status’] == RESPSTATUS_GOOD:
    # Process TLSA RRs
Example query
Fall back

```python
from getdns import *

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB

ext = {'dnssec_return_only_secure': EXTENSION_TRUE}
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    ctx.resolution_type = RESOLUTION_RECURSING
    res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA RRs
```
Example query

Fall back

```python
from getdns import *

cx = Context()
cx.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = cx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    cx.resolution_type = RESOLUTION_RECURSING
    res = cx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA Rrs
```

See also: https://tools.ietf.org/html/draft-ietf-dnsop-dnssec-roadblock-avoidance

And: Discovery method for a DNSSEC validating stub resolver,
    Xavier Torrent Gorjón, University of Amsterdam, July 2015
Example query

Fall back

```python
from getdns import Context
ctx = Context()
ctx.resolution_type = RESOLUTION_STUB
ext = {
    "dnssec_return_only_secure": EXTENSION_TRUE
}
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)
if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    ctx.resolution_type = RESOLUTION_RECURSING
    res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)
if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA RRs
See also: https://tools.ietf.org/html/draft-ietf-dnsop-dnssec-roadblock-avoidance
And: Discovery method for a DNSSEC validating stub resolver, Xavier Torrent Gorjón, University of Amsterdam, July 2015
```
Example query

Fall back

```
from getdns import *
ctx = Context()
ctx.resolution_type = RESOLUTION_STUB
ext = {'dnssec_return_only_secure': EXTENSION_TRUE}
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)
if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    ctx.resolution_type = RESOLUTION_RECURSING
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)
if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA Rrs

See also:
And:

Measurements done at > 8000 RIPE ATLAS probes, +- 10.000 results

64.71% is able to deliver verifiable positive answer
55.67% is able to deliver verifiable negative answer
29.51% is able to deliver verifiable wildcard answer
```
Example query
Fall back

from getdns import *
ctx = Context()
ctx.resolution_type = RESOLUTION_STUB
ext = {
    "dnssec_return_only_secure": EXTENSION_TRUE
}
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)
if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    ctx.resolution_type = RESOLUTION_RECURSING
    res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)
if res is not None and res['status'] == RESPSTATUS_GOOD:
    # Process TLSA Rrs

See also:
And:
Discovery method for a DNSSEC validating stub resolver,
Xavier Torrent Gorjón, University of Amsterdam, July 2015

Query for an A record to echo.v4.nlnetlabs.nl. Server replies with the IP of the recursive resolver!

80% is able to deliver verifiable positive answer
Example query

Fall back

```python
from getdns import *

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    ctx.resolution_type = RESOLUTION_RECURSING
    res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA Rrs

• Roadblock avoidance extension? Nice to have for the nsswitch module!
```
from getdns import *

c = Context()
c.resolution_type = RESOLUTION_STUB

e = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = c.general('443._tcp.getdnsapi.net', RRTYPE_TLSA, e)

if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    c.resolution_type = RESOLUTION_RECURSING
    res = c.general('443._tcp.getdnsapi.net', RRTYPE_TLSA, e)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA Rrs

- Roadblock avoidance extension? Nice to have for the nsswitch module!

- Alternatively bypass DNS network operation completely with:
Example query

Fall back

```
from getdns import *

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB
ext = {'dnssec_return_only_secure': EXTENSION_TRUE}
res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    ctx.resolution_type = RESOLUTION_RECURSING
    res = ctx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA RRs

• Roadblock avoidance extension? Nice to have for the nsswitch module!

• Alternatively, 
```
Example query

Fall back

```python
from getdns import *

cx = Context()
cx.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = cx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    cx.resolution_type = RESOLUTION_RECURSING
    res = cx.general('_443._tcp.getdnsapi.net', RRTYPE_TLSA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA Rrs
```

- Roadblock avoidance extension? Nice to have for the nsswitch module!
- (good application of the dnssec_return_validation_chain_extension extension!)
Example query process records

# Correctly query and process DANE records

```python
if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA RRs
    tlsas = [ answer for reply in res['replies_tree']
                for answer in reply['answer']
                if answer['type'] == RRTYPE_TLSA ]

    # Setup TLS only if the remote certificate (or CA)
    # matches one of the TLSA RRs.

elif res['status'] == RESPSTATUS_ALL_TIMEOUT or \
    res['status'] == RESPSTATUS_ALL_BOGUS_ANSWERS:
    # DON'T EVEN TRY!

else:
    assert(res['status'] == RESPSTATUS_NO_SECURE_ANSWERS)
    # Conventional PKIX without DANE processing
```
C function primitives
Async lookups

getdns_return_t getdns_general(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    getdns_dict *extensions,
    *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);

- **context** contains configuration parameters
  - Stub or recursive modus operandi, timeout values, root-hints, forwarders, trust anchor, search path (+ how to evaluate (not implemented yet) etc.)

- **context** contains the resolver cache (i.e. libunbound context)
C function primitives
Async lookups

```c
getdns_return_t getdns_general(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    getdns_dict *extensions,
    void *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);
```

- context contains configuration parameters
- name and request_type the name and type to lookup
C function primitives
Async lookups

```c
getdns_return_t getdns_general(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    getdns_dict *extensions,
    void *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);
```

- `context` contains configuration parameters
- `name` and `request_type` the name and type to lookup
- `extensions` additional parameters specific for this lookup
  - `return_both_v4_and_v6`, `specify_class`, `dnssec_return_status`, `dnssec_return_only_secure`, `dnssec_return_validation_chain`
  - `add_opt_parameter`
C function primitives
Async lookups

getdns_return_t getdns_general(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    getdns_dict *extensions,
    void *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);

- **context** contains configuration parameters
- **name** and **request_type** the name and type to lookup
- **extensions** additional parameters specific for this lookup
- **userarg** is passed in on the call to **callbackfn**
- **transaction_id** is set to a unique value that is also passed in on the call to **callbackfn**
C function primitives
Async lookups

```c
getdns_return_t getdns_general(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    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
);
// callback_type = complete, cancel, timeout or error
```
C function primitives
Synchronous lookups

getdns_return_t getdns_general(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    getdns_dict *extensions,
    void *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);

getdns_return_t getdns_general_sync(
    getdns_context *context,
    const char *name,
    uint16_t request_type,
    getdns_dict *extensions,
    getdns_dict **response
);

Willem Toorop (NLnet Labs)
C function primitives
Address lookups

```c
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
);
```

- **getdns_address** also lookups in other name systems
  - local files, WINS, mDNS, NIS (only local files implemented)
- **getdns_address** returns both IPv4 and IPv6
  - like when the `return_both_v4_and_v6` extension is set
C function primitives
Reverse lookups

getdns_return_t getdns_hostname(
    getdns_context *context,
    getdns_dict *address,
    getdns_dict *extensions,
    void *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);

- With address: { "address_type": <bindata of "IPv4"> "address_data": <bindata for 185.49.141.37> } will lookup 37.141.49.185.in-addr.arpa PTR
Data structures

typedef struct getdns_dict getdns_dict;
typedef struct getdns_list getdns_list;
typedef struct getdns_bindata { size_t size;
                          uint8_t *data; } getdns_bindata;

• Used to represent extensions, addresses and response objects
Data structures

typedef struct getdns_dict getdns_dict;
typedef struct getdns_list getdns_list;
typedef struct getdns_bindata { size_t size;
                                    uint8_t *data; } getdns_bindata;

- Used to represent extensions, addresses and response objects
- char *getdns_pretty_print_dict(const getdns_dict *dict);

Extension dict

{  
"return_both_v4_and_v6": GETDNS_EXTENSION_TRUE,
"add_opt_parameter":
{ "maximum_udp_payload_size": 1232,
  "do_bit": 1
  "options":
    [ { "option_code": 10
      "option_data": <bindata of 0x96bd16564dfb5f5e > } ] }
typedef struct getdns_dict getdns_dict;
typedef struct getdns_list getdns_list;
typedef struct getdns_bindata { size_t size;
    uint8_t *data; } getdns_bindata;

• Used to represent extensions, addresses and response objects

**Response object dict**

```json
{
    "answer_type": GETDNS_NAME_TYPE_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">
        } ],
    "replies_full": [ <bindata of 0x00008180000100020004...> ],
    "replies_tree": [ { ... first reply ... } ]
}
```
Data structures
Accessor functions

• reading getdns_dicts:

    getdns_return_t getdns_dict_get_dict(
        const getdns_dict *dict, const char *name, getdns_dict **answer);

    getdns_return_t getdns_dict_get_list(
        const getdns_dict *dict, const char *name, getdns_list **answer);

    getdns_return_t getdns_dict_get_bindata(
        const getdns_dict *dict, const char *name, getdns_bindata **answer);

    getdns_return_t getdns_dict_get_int(
        const getdns_dict *dict, const char *name, uint32_t *answer);

    getdns_return_t getdns_dict_get_data_type(
        const getdns_dict *dict, const char *name, getdns_data_type *answer);

    getdns_return_t getdns_dict_get_names(
        const getdns_dict *dict, getdns_list **answer);
Data structures
Accessor functions

- reading `getdns_lists`:

```c
getdns_return_t getdns_list_get_dict(
    const getdns_list *list, size_t index, getdns_dict **answer);

getdns_return_t getdns_list_get_list(
    const getdns_list *list, size_t index, getdns_list **answer);

getdns_return_t getdns_list_get_bindata(
    const getdns_list *list, size_t index, getdns_bindata **answer);

getdns_return_t getdns_list_get_int(
    const getdns_list *list, size_t index, uint32_t *answer);

getdns_return_t getdns_list_get_data_type(
    const getdns_list *list, size_t index, getdns_data_type *answer);

getdns_return_t getdns_list_get_length(
    const getdns_list *this_list, size_t *answer);
```
Data structures
Accessor functions

• Creating/writing to `getdns_dicts`:

```c
getdns_dict * getdns_dict_create();

getdns_return_t getdns_dict_set_dict(
    getdns_dict *dict, const char *name, const getdns_dict *child_dict);

getdns_return_t getdns_dict_set_list(
    getdns_dict *dict, const char *name, const getdns_list *child_list);

getdns_return_t getdns_dict_set_bindata(
    getdns_dict *dict, const char *name, const getdns_bindata
    *child_bindata);

getdns_return_t getdns_dict_set_int(
    getdns_dict *dict, const char *name, uint32_t child_uint32)

void getdns_dict_destroy(getdns_dict *dict);
```
Data structures
Accesser functions

Response object dict

```json
{
    "answer_type": GETDNS_NAME_TYPE_DNS,
    "status": GETDNS_RESP_STATUS_GOOD,
    "canonical_name": `<bindata of "www.getdnsapi.net."`,
    "just address_answers":
    [ { "address_data": `<bindata for 185.49.141.37>`,
        "address_type": `<bindata of "IPv4">`
    }
    ],
    "replies_full": [ `<bindata of 0x00008180000100020004...>` ],
    "replies_tree": [ `{ ... first reply ... }` ],
}
```

```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;
```
Data structures

Accessor functions

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;

• Not so bad in other languages

• Python
  resp = ctx.address('getdnsapi.net')
  addr = resp.just_address_answers[0]["address_data"]

• Nodejs
  function callback(err, resp)
  {
    var addr = resp.just_address_answers[0].address_data;
  }
  ctx.getAddress('getdnsapi.net', callback);
Data structures

Accessor functions

- Not so bad in other languages
- The alternative would introduce a lot of new types:
  - Python:
    ```python
    addr = resp.replies_tree[0]['answer'][0]['rdata']['ipv6_address']
    ```
  - C
    ```c
    getdns_response *resp;  getdns_reply *reply;
    getdns_rrs *rrs;  getdns_rr *rr;
    getdns_rdata *rdata;  struct sockaddr_storage addr;
    if ((r = getdns_response_get_reply(resp, 0, &reply)))
        return r;
    else if ((r = getdns_reply_get_answer_section(reply, &rrs)))
        return r;
    else if ((r = getdns_rrs_get_rr(rrs, &rr)))
        return r;
    else if ((r = getdns_rr_get_rdata(rr, &rdata)))
        return r;
    else if ((r = getdns_rdata_get_rdatafield_address(rdata, 0, &addr)))
        return r;
    ```
Data structures
Accessor functions

• Not so bad in other languages
• The alternative would introduce a lot of new types.
• With current approach, the library can easily grow
• New rdata fields or new extensions without a new API
  (dns cookies, roadblock avoidance, client subnet, etc.)
Data structures
Accessor functions

- Not so bad in other languages
- The alternative would introduce a lot of new types.
- With current approach, the library can easily grow
- New rdata fields or new extensions without a new API (dns cookies, roadblock avoidance, client subnet, etc.)
- Just in time parsing of wireformat data on the roadmap (internally already iterator like accessor types for wireformat data; they will be part of ldns2 too)
Hook into getdns

- Provide function pointers that getdns will use to do memory & IO handling/management
Hook into getdns
Custom memory functions

- Provide function pointers that getdns will use to do memory & IO handling/management

```c
getdns_return_t
getdns_context_create(getdns_context ** context, int set_from_os);

getdns_return_t
getdns_context_create_with_memory_functions(
    getdns_context **context,
    int set_from_os,
    void **malloc) (size_t),
    void **(realloc)(void *, size_t),
    void (**free) (void *)
);
```
Hook into getdns
Custom memory functions

- Provide function pointers that getdns will use to do memory & IO handling/management

```c
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 *)
);
```
Hook into getdns
Custom memory functions

- Provide function pointers that getdns will use to do memory & IO handling/management

```c
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 *)
);

getdns_dict *getdns_dict_create_with_context(  
  getdns_context *context
);
getdns_list *getdns_list_create_with_context(  
  getdns_context *context
);
```
Hook into getdnds
Custom memory functions

- Provide function pointers that getdnds will use to do memory & IO handling/management

```c
getdns_dict *getdns_dict_create_with_context(  
    getdns_context *context
);
getdns_dict *getdns_dict_create_with_memory_functions(  
    void *(*malloc) (size_t),
    void *(*realloc)(void *, size_t),
    void (*free) (void *)
);
getdns_dict *getdns_dict_create_with_extended_memory_functions(  
    void *userarg,
    void *(*malloc) (void *userarg, size_t),
    void *(*realloc)(void *userarg, void *, size_t),
    void (*free) (void *userarg, void *)
);
```
Hook into getdns

Custom event loop

- Poor mans OOP

```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);
```
Hook into getdnds
Custom event loop

- Poor mans OOP

```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);
};
```
Hook into getdns
Custom event loop

- Poor mans OOP

```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);

#define MAX_TIMEOUTS FD_SETSIZE

/* Eventloop based on select */
typedef struct my_eventloop {
    getdns_eventloop base;
    getdns_eventloop_event *fd_events[FD_SETSIZE];
    uint64_t fd_timeout_times[FD_SETSIZE];
    getdns_eventloop_event *timeout_events[MAX_TIMEOUTS];
    uint64_t timeout_times[MAX_TIMEOUTS];
} my_eventloop;

my_eventloop my_loop;
getdns_context_set_eventloop(context, &my_loop.base)
```

User program
Hook into getdns
Custom event loop

• Poor mans OOP

```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
)

#define MAX_TIMEOUTS FD_SETSIZE

/* Eventloop based on select */
typedef struct my_eventloop {
    getdns_eventloop base;
    getdns_eventloop_event *fd_events[FD_SETSIZE];
    uint64_t fd_timeout_times[FD_SETSIZE];
    getdns_eventloop_event *timeout_events[MAX_TIMEOUTS];
    uint64_t timeout_times[MAX_TIMEOUTS];
} my_eventloop;

my_eventloop my_loop;
getdns_context_set_eventloop(context, &my_loop.base)
```

Timeouts must be a set that may be modified during iteration
Hook into getdnts
Custom event loop

#define MAX_TIMEOUTS FD_SETSIZE

/* Eventloop based on select */
typedef struct my_eventloop {
  getdns_eventloop base;
  getdns_eventloop_event *fd_events[FD_SETSIZE];
  uint64_t fd_timeout_times[FD_SETSIZE];
  getdns_eventloop_event *timeout_events[MAX_TIMEOUTS];
  uint64_t timeout_times[MAX_TIMEOUTS];
} my_eventloop;

void my_eventloop_init(my_eventloop *loop)
{
  static getdns_eventloop_vmt my_eventloop_vmt = {
    my_eventloop_cleanup,
    my_eventloop_schedule, my_eventloop_clear, NULL, NULL
  };

  (void) memset(loop, 0, sizeof(my_eventloop));
  loop->base.vmt = &my_eventloop_vmt;
}

my_eventloop my_loop;
my_eventloop_init(&my_loop);
getdns_context_set_eventloop(context, &my_loop.base)
Hook into getdns
Custom event loop

- From specification section 1.8:
  ... *Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used.*

- libevent

```c
#include <getdns/getdns_ext_libevent.h>
#include <getdns/getdns_ext_libevent.h>

my_eventloop_init(&my_loop);

my_eventloop_init(&my_loop);

getdns_extension_set_libevent_base(context, &my_loop.base);
getdns_extension_set_libevent_base(context, &my_loop.base);

struct event_base *base = event_base_new();
struct event_base *base = event_base_new();

getdns_extension_set_libevent_base(context, base);
getdns_extension_set_libevent_base(context, base);

getdns_address(context, "getdnsapi.net", 0, 0, 0, callback);
getdns_address(context, "getdnsapi.net", 0, 0, 0, callback);

event_base_dispatch(base);
event_base_dispatch(base);

event_base_free(base);
event_base_free(base);

getdns_context_set_eventloop(context, &my_loop.base);
getdns_context_set_eventloop(context, &my_loop.base)
```
#define MAX_TIMEOUTS FD_SETSIZE

/* Eventloop based on select */
typedef struct my_eventloop {
  getdns_eventloop base;
  getdns_eventloop_event *fd_events [FD_SETSIZE];
  uint64_t fd_timeout_times [FD_SETSIZE];
  getdns_eventloop_event *timeout_events [MAX_TIMEOUTS];
  uint64_t timeout_times [MAX_TIMEOUTS];
} my_eventloop;

void my_eventloop_init (my_eventloop *loop) {
  static getdns_eventloop_vmt my_eventloop_vmt = {
    my_eventloop_cleanup,
    my_eventloop_schedule,
    my_eventloop_clear,
    NULL,
    NULL};
  (void) memset (loop, 0, sizeof (my_eventloop));
  loop->base.vmt = &my_eventloop_vmt;
}

my_eventloop my_loop;
my_eventloop_init (&my_loop);
getdns_context_set_eventloop (context, &my_loop.base)

Hook into getdns
Custom event loop

- libevent
  Include : #include <getdns/getdns_ext_libevent.h>
  Use : getdns_extension_set_libevent_base (context, base);
  Link : -lgetdns -lgetdns_ext_event

- libev
  Include : #include <getdns/getdns_ext_libev.h>
  Use : getdns_extension_set_libev_loop (context, loop);
  Link : -lgetdns -lgetdns_ext_ev

- libuv
  Include : #include <getdns/getdns_ext_libuv.h>
  Use : getdns_extension_set_libuv_loop (context, base);
  Link : -lgetdns -lgetdns_ext_uv

User program
Hook into getdnds
Custom event loop

User program

• Destructor, called on
  – getdns_context_destroy()
  – getdns_context_detach_eventloop()
  – getdns_context_set_eventloop()
Hook into getdns

Custom event loop

<getdns_extra.h>

```c
/* 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;

getdns_return_t my_eventloop_schedule(getdns_eventloop *loop,
                                       int fd, uint64_t timeout, getdns_eventloop_event *event)
{
    my_eventloop *my_loop = (my_eventloop *)loop;

    assert(loop);
    assert(event);
    assert(fd < FD_SETSIZE);

    if (fd >= 0 && (event->read_cb || event->write_cb)) {
        assert(my_loop->fd_events[fd] == NULL);
    }
```
Hook into getdnds
Custom event loop

```c
/* 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;

getdns_return_t my_eventloop_schedule(getdns_eventloop *loop,
    int fd, uint64_t timeout, getdns_eventloop_event *event)
{
    my_eventloop *my_loop = (my_eventloop *)loop;

    if (fd >= 0 && (event->read_cb || event->write_cb)) {
        my_loop->fd_events[fd] = event;
        my_loop->fd_timeout_times[fd] = get_now_plus(timeout);
        event->ev = (void *) (intptr_t) fd + 1;
        return GETDNS_RETURN_GOOD;
    }
}
```

User program
Hook into getdns

Custom event loop

```c
getdns_return_t my_eventloop_schedule(getdns_eventloop *loop,
    int fd, uint64_t timeout, getdns_eventloop_event *event)
{
    my_eventloop *my_loop = (my_eventloop *)loop;

    if (fd >= 0 && (event->read_cb || event->write_cb)) {
        my_loop->fd_events[fd] = event;
        my_loop->fd_timeout_times[fd] = get_now_plus(timeout);
        event->ev = (void *) (intptr_t) fd + 1;
        return GETDNS_RETURN_GOOD;
    }

    assert(event->timeout_cb && !event->read_cb && !event->write_cb);
    for (size_t i = 0; i < MAX_TIMEOUTS; i++) {
        if (my_loop->timeout_events[i] == NULL) {
            my_loop->timeout_events[i] = event;
            my_loop->timeout_times[i] = get_now_plus(timeout);
            event->ev = (void *) (intptr_t) i + 1;
            return GETDNS_RETURN_GOOD;
        }
    }

    return GETDNS_RETURN_GENERIC_ERROR;
}
```

User program
Hook into getdns
Custom event loop

getdns_return_t
my_eventloop_clear(getdns_eventloop *loop, getdns_eventloop_event *event)
{
    my_eventloop *my_loop = (my_eventloop *)loop;
    size_t i;

    i = (intptr_t)event->ev - 1;

    if (event->timeout_cb && !event->read_cb && !event->write_cb) {
        my_loop->timeout_events[i] = NULL;
    } else {
        my_loop->fd_events[i] = NULL;
    }

    event->ev = NULL;
    return GETDNS_RETURN_GOOD;
}
Hook into getdns

Custom event loop

Running the loop

User program

```c
uint64_t now, timeout = (uint64_t)-1;
size_t i;

now = get_now_plus(0);

for (i = 0; i < MAX_TIMEOUTS; i++) {
    if (!my_loop->timeout_events[i])
        continue;

    if (now > my_loop->timeout_times[i])
        my_timeout_cb(my_loop->timeout_events[i]);

    else if (my_loop->timeout_times[i] < timeout)
        timeout = my_loop->timeout_times[i];
}
```
Hook into getdns

Custom event loop

User program

```c
fd_set readfds, writefds;
int fd, max_fd = -1;

FD_ZERO(&readfds);
FD_ZERO(&writefds);

for (fd = 0; fd < FD_SETSIZE; fd++) {
    if (!my_loop->fd_events[fd])
        continue;

    if (my_loop->fd_events[fd]->read_cb)
        FD_SET(fd, &readfds);
    if (my_loop->fd_events[fd]->write_cb)
        FD_SET(fd, &writefds);

    if (fd > max_fd)
        max_fd = fd;

    if (my_loop->fd_timeout_times[fd] < timeout)
        timeout = my_loop->fd_timeout_times[fd];
}
if (max_fd == -1 && timeout == (uint64_t)-1)
    return;
```
Hook into getdns
Custom event loop

Running the loop

User program

```c
struct timeval tv;

if (now > timeout) {
    tv.tv_sec = 0;
    tv.tv_usec = 0;
} else {
    tv.tv_sec = (timeout - now) / 1000000;
    tv.tv_usec = (timeout - now) % 1000000;
}
if (select(max_fd + 1, &readfds, &writefds, NULL, &tv) < 0) {
    perror("select() failed");
    exit(EXIT_FAILURE);
}
```
Hook into getdnds

Custom event loop

Running the loop

```c
now = get_now_plus(0);
for (fd = 0; fd < FD_SETSIZE; fd++) {
    if (my_loop->fd_events[fd] &&
        my_loop->fd_events[fd]->read_cb &&
        FD_ISSET(fd, &readfds))
        my_read_cb(fd, my_loop->fd_events[fd]);

    if (my_loop->fd_events[fd] &&
        my_loop->fd_events[fd]->write_cb &&
        FD_ISSET(fd, &writefds))
        my_write_cb(fd, my_loop->fd_events[fd]);

    if (my_loop->fd_events[fd] &&
        my_loop->fd_events[fd]->timeout_cb &&
        now > my_loop->fd_timeout_times[fd])
        my_timeout_cb(my_loop->fd_events[fd]);

    i = fd;
    if (my_loop->timeout_events[i] &&
        my_loop->timeout_events[i]->timeout_cb &&
        now > my_loop->timeout_times[i])
        my_timeout_cb(my_loop->timeout_events[i]);
}
```
Hook into getdns
Custom event loop

```javascript
var getdns = require('getdns');

function callback(err, result) {
    console.log(err ? Err : result.canonical_name + ': ' + JSON.stringify(result.just_address_answers));
}
ctx = getdns.createContext();
ctx.getAddress('getdnsapi.net', callback);
ctx.getAddress('verisignlabs.com', callback);
ctx.getAddress('sinodun.com', callback);
ctx.getAddress('nomountain.net', callback);
ctx.getAddress('vbsdcon.com', callback);
```

Program output:

```
willem@bonobo:~/vbsdcon$ nodejs parallel.js
getdnsapi.net.: [{"address_data":[42,4,185,0,0,0,1,0,0,0,0,0,0,0,0,55], ... sinodun.com.: [{"address_data":[88,98,24,67],"address_type":"IPv4"}] vbsdcon.com.: [{"address_data":[69,58,186,114],"address_type":"IPv4"}] verisignlabs.com.: [{"address_data":[38,32,0,116,0,19,68,0,0,0,0,0,0,0,2 ... nomountain.net.: [{"address_data":[38,7,242,152,0,5,16,75,0,0,0,0,11,128 ...```
Roadmap

• Current release 0.3.3
• More bindings (ruby (alpha), perl, lua, go (proposed))
• More platforms (windows, android)
• Before 1.0 (this year)
  – No more dependency on ldns
  – Just-in-time parsing of response objects
  – The complete spec implemented
    • add_warning_for_bad_dns & add_call_debugging extensions
    • TSIG
• After 1.0
  – Multi-threading & multi-processes support
  – statefull session reuse
Security starts with a name

getdns
Unbound security

website  https://getdnsapi.net
API spec  https://getdnsapi.net/spec.html
latest tarball  https://getdnsapi.net/dist/getdns-0.3.3.tar.gz
github repo  https://github.com/getdnsapi/getdns
node repo  https://github.com/getdnsapi/getdns-node
python repo  https://github.com/getdnsapi/getdns-python-bindings
java repo  https://github.com/getdnsapi/getdns-java-bindings
php repo  https://github.com/getdnsapi/getdns-php-bindings
API list  https://getdnsapi.net/mailman/listinfo/spec
users list  https://getdnsapi.net/mailman/listinfo/users
me  Willem Toorop <willem@nlnetlabs.nl>