Feb 01, 2025 - FOSDEM

getaddrinfo sucks Everything else is much worse

Mozilla

Valentin Gosu

Senior Staff Engineer, Firefox Networking team (Necko)

Native of 🇷🇴 residing in 🇸🇪

valentin@mozilla.com

https://fosstodon.org/@valenting

I 💖 emoji.

Natural Introvert. Please come say hi. 🙂

Full disclosure

01 Yes, the title is clickbait

02 No, getaddrinfo doesn’t really suck

03 Unless you want to use it for something it can’t do.

04 This talk may include rants

05 I rarely know what I’m doing 🤪

01

getaddrinfo

Timeline of getaddrinfo

Protocol Independent Interfaces, IEEE Std 1003.1g DRAFT 6.3

November 1995

RFC 2133 - Basic Socket Interface Extensions for IPv6

April 1997

RFC 2553

March 1999

RFC 3493

February 2003

getaddrinfo

```
int getaddrinfo(const char *nodename, const char *servname,
                const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *ai);

struct addrinfo {
  int     ai_flags;     /* AI_PASSIVE, AI_CANONNAME,
                           AI_NUMERICHOST, .. */
  int     ai_family;    /* AF_xxx */
  int     ai_socktype;  /* SOCK_xxx */
  int     ai_protocol;  /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
  socklen_t  ai_addrlen;   /* length of ai_addr */
  char   *ai_canonname; /* canonical name for nodename */
  struct sockaddr  *ai_addr; /* binary address */
  struct addrinfo  *ai_next; /* next structure in linked list */
};
```

What’s great about getaddrinfo?

1. It’s simple
2. It works!
3. It’s well documented
4. It’s available on all platforms
5. It works (roughly) the same on all platforms

What sucks about getaddrinfo?

1. It’s a synchronous API
2. It is very limited
3. It doesn’t expose full response details (TTL, EDNS0, rcode, CNAME chain)
4. Minor implementation differences do exist
5. No way to control caching or use of /etc/hosts

How getaddrinfo is used in Firefox

- Thread pool for calling getaddrinfo
- nsHostResolver has request queues with different priorities.
- Each thread checks the queues in order and calls getaddrinfo
- Caching (Firefox DNS cache, OS DNS cache, ISP DNS cache)
- TTLs
- DNSQuery_A on Windows (only for getting TTL values)

Enter DNS over HTTPS (RFC 8484)

- Firefox needed a DNS packet parser for DoH
- No longer sync - just one thread that polls the socket and parses responses
- We now have TTL values
- Can parse different responses such as CNAME, TXT, OPT
- SVCB/HTTPS RR

- Trusted Recursive Resolver (TRR) vs DNS over HTTPS (DoH)

SVCB/HTTPS Resource Records (RFC 9460)

User tries to load http://example.com

```
$ORIGIN example.com.
www  7200 IN HTTPS 1 . alpn=h2
www  7200 IN HTTPS 2 . alpn=h3 port=8443
```

This host has a HTTPS record:

- No plain-text HTTP.
- The host supports HTTP/2 on the default port
- For HTTP/3, we can connect to port 8443
    - the URL bar will still show https://www.example.com (note that the origin will still be www.example.com:443 - default HTTPS port).

ipv4hint, ipv6hint, ech

What about non DoH clients?

Why use native APIs for HTTPS RR

- using the same resolver as the rest of the OS
- OS cache
- Easier implementation?

Expectations for native APIs

✅Exists
✅Well documented
✅Actually works

Linux: res_nquery

✅Exists
✅Well documented
✅Actually works

```
int res_ninit(res_state statep);
void res_nclose(res_state statep);

int res_nquery(res_state statep,
               const char *dname, int class, int type,
               unsigned char answer[.anslen], int anslen);
```

Android: res_query

Android NDK DNS resolution with libresolv

Asked 4 years, 5 months ago Modified 3 years, 4 months ago Viewed 1k times Part of Mobile Development Collecti

1

I want to use an existing C/C++ library in an Android project. The library contains calls to `res_query` in the `resolv.h` header (libresolv on Linux), which exists in the NDK and, according to Wikipedia, is part of Android's Bionic libc.

The program builds and links fine after removing the explicit link instruction `-lresolv` from the build file, but DNS resolution with `res_query` always fails (returns `-1`).

To further inspect this issue, I built an example program using only a single call to `res_query` using the example C++ project of Android Studio:

1 Answer Sorted by: Highest score (default)

0

res_query is not supported in API level 29 and above. You can replace **res_query** by **android_res_nquery** and **android_res_nresult**

Check https://developer.android.com/ndk/reference/group/networking for more details

Share Improve this answer Follow

answered Aug 17, 2021 at 15:40

Aloui Mohamed Khalil

1 ● 1

✅Exists
❓Well documented
⛔Actually works

Android: android_res_nquery

```
void* handle = dlopen("libandroid_net.so", RTLD_LAZY);
auto android_res_nquery = (int (*)(...)) dlsym(handle, "android_res_nquery");
int resp_handle = android_res_nquery(0, "google.com", ns_c_in, 65, 0);

struct pollfd fds;
fds.fd = resp_handle;
fds.events = POLLIN; // Wait for read events
// Wait for an event on the file descriptor
int ret = poll(&fds, 1, -1); // -1 means no timeout

 char answer[3200];
 ssize_t bytes_received = recv(resp_handle, answer, 3200 - 1, 0);

// First 8 bytes in answer are the UDP header.
```

Android: android_res_nquery

Only available on Android 10+

What I like about it:

- Can specify caching (does it work? 🤷)
- Can specify network interface

```
ResNsendFlags{
  ANDROID_RESOLV_NO_RETRY = 1 << 0,
  ANDROID_RESOLV_NO_CACHE_STORE = 1 << 1,
  ANDROID_RESOLV_NO_CACHE_LOOKUP = 1 << 2
}
android_res_nquery(net_handle_t network, const char *dname, int ns_class,
int ns_type, uint32_t flags)
```

✅Exists
✅Well documented
🤞Actually works

Windows: DNSQuery_A

```
DNS_STATUS DnsQuery_A(
  [in]                PCSTR      pszName,
  [in]                WORD       wType,
  [in]                DWORD      Options,
  [in, out, optional] PVOID      pExtra,
  [out, optional]     PDNS_RECORD *ppQueryResults,
  [out, optional]     PVOID      *pReserved
);
```

```
PDNS_RECORD result = nullptr;

DNS_STATUS status = DnsQuery_A(
        host.c_str(),
        65, // Type 65 corresponds to HTTPS record
        DNS_QUERY_STANDARD,
        nullptr, &result, nullptr
);
if (status != ERROR_SUCCESS) {
        LOG("DnsQuery_A failed with error: %ld", status);
        return false;
}
LOG("result %p", result); // This is NULL on Windows 10! 😭
```

✅Exists
✅Well documented
🤷Actually works

Windows 10 HTTPS RR bug: https://aka.ms/AAo4y7b
Please upvote 👍 (if you're on Windows)

Windows: DNSQuery_A

```
73    PDNS_RECORDA dnsData = nullptr;
74    DNS_STATUS status = DnsQuery_A(aHost.BeginReading(), reqFamily, aFlags,
75                                   nullptr, &dnsData, nullptr);
76    if (status == DNS_INFO_NO_RECORDS || status == DNS_ERROR_RCODE_NAME_ERROR ||
77        !dnsData) {
78      LOG("No DNS records found for %s. status=%lX. reqFamily = %X\n",
79          aHost.BeginReading(), status, reqFamily);
80      return NS_ERROR_FAILURE;
81    } else if (status != NOERROR) {
82      LOG_WARNING("DnsQuery_A failed with status %lX.\n", status);
83      return NS_ERROR_UNEXPECTED;
84    }
85
86    for (PDNS_RECORDA curRecord = dnsData; curRecord;
87         curRecord = curRecord->pNext) {
88      // Only records in the answer section are important
89      if (curRecord->Flags.S.Section != DnsSectionAnswer) {
90        continue;
91      }
92      if (curRecord->wType != reqFamily) {
93        continue;
94      }
95
96      aCallback(curRecord);
97    }
98
99    DnsFree(dnsData, DNS_FREE_TYPE::DnsFreeRecordList);
100   return NS_OK;
```

DNS query options

Expand table

Constant	Value	Meaning
DNS_QUERY_STANDARD	0x00000000	Standard query.
DNS_QUERY_ACCEPT_TRUNCATED_RESPONSE	0x00000001	Returns truncated results. Does not retry under TCP.
DNS_QUERY_USE_TCP_ONLY	0x00000002	Uses TCP only for the query.
DNS_QUERY_NO_RECURSION	0x00000004	Directs the DNS server to perform an iterative query (specifically directs the DNS server not to perform recursive resolution to resolve the query).
DNS_QUERY_BYPASS_CACHE	0x00000008	Bypasses the *resolver* cache on the lookup.
DNS_QUERY_NO_WIRE_QUERY	0x00000010	Directs DNS to perform a query on the local cache only.**Windows 2000 Server and Windows 2000 Professional:** This value is not supported. For similar functionality, use **DNS_QUERY_CACHE_ONLY**.
DNS_QUERY_NO_LOCAL_NAME	0x00000020	Directs DNS to ignore the local name.**Windows 2000 Server and Windows 2000 Professional:** This value is not supported.
DNS_QUERY_NO_HOSTS_FILE	0x00000040	Prevents the DNS query from consulting the HOSTS file.**Windows 2000 Server and Windows 2000 Professional:** This value is not supported.
DNS_QUERY_NO_NETBT	0x00000080	Prevents the DNS query from using NetBT for resolution.**Windows 2000 Server and Windows 2000 Professional:** This value is not supported.
DNS_QUERY_WIRE_ONLY	0x00000100	Directs DNS to perform a query using the network only, bypassing local information.**Windows 2000 Server and Windows 2000 Professional:** This value is not supported.
DNS_QUERY_RETURN_MESSAGE	0x00000200	Directs DNS to return the entire DNS response message.**Windows 2000 Server and Windows 2000 Professional:** This value is not supported.
DNS_QUERY_MULTICAST_ONLY	0x00000400	Prevents the query from using DNS and uses only Local Link Multicast Name Resolution (LLMNR).**Windows Vista and Windows Server 2008 or later.:** This value is supported.
DNS_QUERY_NO_MULTICAST	0x00000800	
DNS_QUERY_TREAT_AS_FQDN	0x00001000	Prevents the DNS response from attaching suffixes to the submitted

```
    DNS_TLSA_DATA       Tlsa;
    DNS_SVCB_DATA       SVCB;
    DNS_SVCB_DATA       Svcb;
    DNS_UNKNOWN_DATA    UNKNOWN;
    DNS_UNKNOWN_DATA    Unknown;
    PBYTE               pDataPtr;
  } Data;
} DNS_RECORDA, *PDNS_RECORDA;
```

```
typedef struct {
  IP4_ADDRESS IpAddress;
} DNS_A_DATA, *PDNS_A_DATA;
```

```
nsresult rv = DNSPacket::ParseHTTPS(current->wDataLength, parsed, 0,
                                    ptr, current->wDataLength, aHost);
```

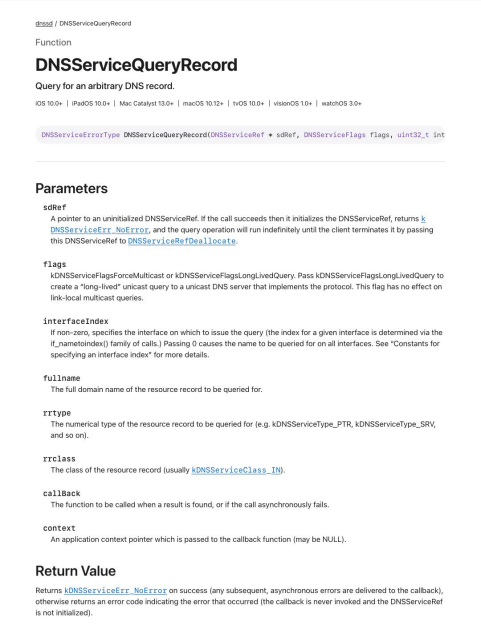
MacOS: res_query

Bug 1882856 - Crash in [@ dns_res_send] with "network.dns.native_https_query" on MacOS

Only use this if your application is single threaded. This is not thread safe.

✅Exists
❓Well documented
🙃Actually works

MacOS: DNSServiceQueryRecord



dnssd / DNSServiceQueryRecord

Function

DNSServiceQueryRecord

Query for an arbitrary DNS record.

iOS 10.0+ | iPadOS 10.0+ | Mac Catalyst 13.0+ | macOS 10.12+ | tvOS 10.0+ | visionOS 1.0+ | watchOS 3.0+

```
DNSServiceErrorType DNSServiceQueryRecord(DNSServiceRef * sdRef, DNSServiceFlags flags, uint32_t int
```

Parameters

`sdRef`
A pointer to an uninitialized DNSServiceRef. If the call succeeds then it initializes the DNSServiceRef, returns kDNSServiceErr_NoError, and the query operation will run indefinitely until the client terminates it by passing this DNSServiceRef to DNSServiceRefDeallocate.

`flags`
kDNSServiceFlagsForceMulticast or kDNSServiceFlagsLongLivedQuery. Pass kDNSServiceFlagsLongLivedQuery to create a "long-lived" unicast query to a unicast DNS server that implements the protocol. This flag has no effect on link-local multicast queries.

`interfaceIndex`
If non-zero, specifies the interface on which to issue the query (the index for a given interface is determined via the if_nametoindex() family of calls.) Passing 0 causes the name to be queried for on all interfaces. See "Constants for specifying an interface index" for more details.

`fullname`
The full domain name of the resource record to be queried for.

`rrtype`
The numerical type of the resource record to be queried for (e.g. kDNSServiceType_PTR, kDNSServiceType_SRV, and so on).

`rrclass`
The class of the resource record (usually kDNSServiceClass_IN).

`callBack`
The function to be called when a result is found, or if the call asynchronously fails.

`context`
An application context pointer which is passed to the callback function (may be NULL).

Return Value

Returns kDNSServiceErr_NoError on success (any subsequent, asynchronous errors are delivered to the callback), otherwise returns an error code indicating the error that occurred (the callback is never invoked and the DNSServiceRef is not initialized).

✅Exists
🤔Well documented

```
void queryRecordCallback(DNSServiceRef sdRef, DNSServiceFlags flags,  uint32_t interfaceIndex,
DNSServiceErrorType errorCode, const char *fullname, uint16_t rrtype, uint16_t rrclass,
uint16_t rdlen, const void *rdata, uint32_t ttl, void *context
) {
        if (errorCode == kDNSServiceErr_NoError) {
            printf("Received record for: %s\n", fullname);
        } else {
            printf("Error querying record: %d\n", errorCode);
        }
}

DNSServiceRef sdRef;
DNSServiceErrorType error = DNSServiceQueryRecord(&sdRef, 0, 0, “google.com”,
        1, kDNSServiceClass_IN, queryRecordCallback, nullptr);
if (error != kDNSServiceErr_NoError) {
        printf("Error starting query: %d\n", error);
        return;
}
DNSServiceProcessResult(sdRef); // Calls the callback
DNSServiceRefDeallocate(sdRef);
```

MacOS: DNSServiceQueryRecord

Bug 1941128 - Tabs hang in DNS lookups on macOS after bug 1938293

If the domain/HTTPS RR doesn't exist, ***the callback never gets called***! 🤯🤦‍♂️😱

WAT?!?

✅Exists
👎Well documented
😱Actually works

MacOS: DNSServiceQueryRecord

```
void queryRecordCallback(...)
) {
        if (errorCode == kDNSServiceErr_NoError) {
            printf("Received record for: %s\n", fullname);
        } else {
            printf("Error querying record: %d\n", errorCode);
        }
}

DNSServiceRef sdRef;
DNSServiceErrorType error =
DNSServiceQueryRecord(&sdRef, 0, 0, host.c_str(),
        1, kDNSServiceClass_IN, queryRecordCallback,
nullptr);
if (error != kDNSServiceErr_NoError) {
        printf("Error starting query: %d\n", error);
        return;
}
```

```
int fd = DNSServiceRefSockFD(sdRef);
fd_set readfds;
FD_ZERO(&readfds);
FD_SET(fd, &readfds);

struct timeval timeout;
timeout.tv_sec = 30;  // 30-second timeout
timeout.tv_usec = 0;

int result = select(fd + 1, &readfds, NULL, NULL, &timeout);
if (result > 0 && FD_ISSET(fd, &readfds)) {
        // Process the result
        DNSServiceProcessResult(sdRef);
} else if (result == 0) {
        printf("select() timed out\n");
} else if (result < 0) {
        printf("select() failed\n");
}
DNSServiceRefDeallocate(sdRef);
```

Lessons learned

- The API capabilities vary widely across platforms.
- An abstraction library would be nice.
- Sometimes implementing your own DNS client might be worth it. Or use an existing one.
- When using these APIs document & blog about your experience.

What API would I like?

- DnsQuery_A/Ex - lots of options for caching, /etc/hosts, network, protocol
- android_res_query - access to the socket and response bytes
- Better control over request bytes as well.
- Like getaddrinfo - **I’d like something that works well everywhere**!

Thank you