1. Test Setup

NetX Validation System is a platform that executes Test Scenarios. Each Test Scenario is a NetX application that targets a specific mDNS functionality or a group of related functionalities. A test scenario also contains a peer application that interacts with the test application.

NetX Duo library and mDNS library source files must be compiled with the following definitions in order to exercise all the test cases.

/\* Enable mDNS over IPv6 feature. \*/

#define NX\_MDNS\_ENABLE\_IPV6

/\* Enable IPv6 multicast feature to support mDNS over IPv6 network \*/

#define NX\_IPV6\_MULTICAST\_ENABLE

/\* Enable IPv6 address change notify feature to support mDNS over IPv6 network \*/

#define NX\_ENABLE\_IPV6\_ADDRESS\_CHANGE\_NOTIFY

/\* Enable dual interface support in order to test mDNS on a multi-home system. \*/

#define NX\_MAX\_PHYSICAL\_INTERFACES 2

/\* Enable optional mDNS address check feature \*/

#define NX\_MDNS\_ENABLE\_ADDRESS\_CHECK

/\* Enable optional mDNS POOF feature \*/

#define NX\_MDNS\_ENABLE\_CLIENT\_POOF

/\* Enable optional feature that allows mDNS to monitor network traffic for negative response\*/

#define NX\_MDNS\_ENABLE\_SERVER\_NEGATIVE\_RESPONSES

Make sure the symbol NX\_DISABLE\_ERROR\_CHECKING is not defined.

The default size of local and peer cache in following tests is 5120 bytes.

1. Test Scenarios

This section describes NetX Test Scenarios.

* 1. Internal function Test (netx\_mdns\_internal\_function\_test.c). This scenario verifies that the internal functions on Resource Record (RR) cache process are able to handle insertion and deletion operations of RR and strings.

Test Procedure:

1. Create mDNS instance.
2. Set cache notify callback.
3. Repeat adding strings to local cache till the option fails.
4. Check the local cache state. It should be full.
5. Delete strings from local cache that would cause the memory in the cache to fragment.
6. Add a string that fits the free area of cache but since the free area is not continuous, the string cannot be added.
7. Check the cache state. It should be fragmented.
8. Repeat (3) to (7) to test peer cache.
9. Delete and recreate mDNS instance.
10. Create 5 threads to randomly add RRs into local cache.
11. Delete all inserted RR in threads.
12. Repeat (10) to (11) to test peer cache.
13. Delete and recreate mDNS instance.
14. Insert RR into local cache until cache full
15. Check the count of inserted RR.
16. Delete and recreate mDNS instance.
17. Insert 2 RRs into cache. Then delete the first RR to bring an empty slot in local cache.
18. Insert the 3rd RR into cache. Check whether this RR is inserted into the empty slot.
19. Repeat (13) to (18) to test peer cache.
20. Repeat (13) to (18), Use string instead of RR to test local cache.
21. Repeat (13) to (18), Use string instead of RR to test peer cache.
22. Delete and recreate mDNS instance.
23. Insert 3 strings into local cache.
24. Delete the string in middle.
25. Insert a string that is less than the deleted string
26. Check the tail
27. Delete all strings and delete mDNS instance.
28. Verify that after this test scenario, no error should occur.
    1. Create and delete Test (netx\_mdns\_create\_delete\_test.c). This scenario verifies that nx\_mdns\_create and nx\_mdns\_delete APIs will not crash and instead they return the error status when there are duplicate creation or deletion.

Test Procedure:

1. Create mDNS instance.
2. Create mDNS instance again. Check the return status should not be success.
3. Repeat (2) five times.
4. Delete mDNS instance.
5. Delete mDNS instance again. Check the return status should not be success.
6. Repeat (5) five times.
7. Create mDNS instance and let the system idle five seconds. Then delete mDNS instance.
8. Create mDNS instance and delete it immediately.
9. Repeat (8) five times.
10. Verify that after this test scenario, no error occurs.
    1. One shot query Test (netx\_mdns\_one\_shot\_query\_test.c). This scenario verifies that one shot query stops sending query after one response is received. Page 8, Section 5.1, RFC 6762.

Compilation Options: #define NX\_MDNS\_ENABLE\_NEGATIVE\_RESPONSES

Test Procedure:

1. Create mDNS instance.
2. Add a one shot query.
3. On receiving the first two queries, do nothing. On receiving the third query, inject a response.
4. Check whether or not more queries are sent.
5. Add a one shot query.
6. Check whether or not more queries are sent.
7. Verify that after this test scenario, no error occurs.
   1. Local cache continuous query Test (netx\_mdns\_local\_cache\_continuous\_query\_test.c). This scenario verifies when the answer is in local cache, the continuous query should add this record as known answer.

Test Procedure:

1. Create mDNS instance.
2. Wait for host probing.
3. Add a service. Service name is “test.\_http.\_tcp.local”.
4. Wait for service probing.
5. Start a continuous query with “http.\_tcp” type.
6. Check the continuous query should include the local service “test.\_http.\_tcp.local”.
7. Verify that after this test scenario, no error should occur.
   1. Local cache one shot query Test (netx\_mdns\_local\_cache\_one\_shot\_query\_test.c). This scenario verifies when the answer is in local cache, one shot query can get the service correctly. And the content of service is checked.

Test Procedure:

1. Create mDNS instance.
2. Add a service. Service name is “ARMMDNSTest.\_ipp.\_tcp.local”. Service text is “paper=A4”.
3. Start a one shot query.
4. Check the return status should be NX\_SUCCESS.
5. Check every field of service. Interface index, service\_domain, service\_host, service\_ipv4, service\_name, service\_port, service\_priority, service\_text, service\_text\_valid, service\_type and service\_weight.
6. Verify that after this test scenario, no error should occur.
   1. Service lookup Test (netx\_mdns\_service\_lookup\_test.c). This scenario verifies that nx\_mdns\_service\_lookup API works as expected with different parameters.

Test Procedure:

1. Create mDNS instance and add a full service. The service name is “CanonMF4500w.\_http.\_tcp.local”.
2. Lookup the service with type “\_http.\_tcp” and index 0.
3. Lookup the service with type“\_http1.\_tcp” and index 0.
4. Lookup the service with with type “\_http.\_tcp” and index from 1 to 1000.
5. Lookup the service with name “CanonMF4500w”, type “\_http.\_tcp” and index 0.
6. Lookup the service with name “CanonMF4500w1”, type “\_http.\_tcp” and index 0.
7. Lookup the service with right name “CanonMF4500w”, type “\_http.\_tcp” and index 0 from 1 to 1000.
8. Lookup the service with index 0.
9. Lookup the service with index from 1 to 1000.
10. Verify that after this test scenario, no error occurs.
    1. Service add and delete Test (netx\_mdns\_service\_add\_delete\_test.c). This scenario verifies that nx\_mdns\_service\_add and nx\_mdns\_service\_delete APIs work as expected in each situation.

Test Procedure:

1. Create mDNS instance.
2. Add a service and clear cache.
3. Check no RR exists in local cache.
4. Add a service and check the RR count in local cache.
5. Add the same service as (4) in local cache and check the return status. The return status should not be success.
6. Add the second service with sub type and check the RR count in local cache.
7. Add the third service and check the RR count in local cache.
8. Add the fourth service and check the RR count in local cache.
9. Add the fifth service and check the RR count in local cache.
10. Delete the first service and check the RR count in local cache.
11. Delete the second service and check the RR count in local cache.
12. Delete the third service and check the RR count in local cache.
13. Delete the fourth service and check the RR count in local cache.
14. Delete the fifth service and check the RR count in local cache.
15. Enable interface of mDNS which will add one RR into local cache. Check the RR count in the local cache.
16. Repeat (4) to (13).
17. Verify that after this test scenario, no error occurs.
    1. Announcement repeat Test(netx\_mdns\_annoucement\_repeat\_test.c). This scenario verifies that the parameters in repeated announcement works as expected.

Test Procedure:

1. Create mDNS instance and add a service.
2. Set all parameters in announcement to default. That is,
   1. The initial interval between two cycles is 100 ticks.
   2. In each cycle, announcement is repeated once.
   3. The factor of interval between two cycles is 1.
   4. The interval of two announcements in the same cycle is 0.
   5. The max interval between two cycles is 0xFFFFFFFF ticks.
   6. The cycle is repeated 3 times.
3. Add a service and check the elapsed time between consecutive announcements.
4. Check whether all announcements are sent.
5. Delete and recreate a service.
6. Change the parameter of (a) from 100 to 200 in (2).
7. Repeat (3) to (4).
8. Delete and recreate a service.
9. Change the parameter of (b) from 1 to 2 in (2).
10. Repeat (3) to (4).
11. Delete and recreate a service.
12. Change the parameter of (c) from 1 to 2 in (2).
13. Repeat (3) to (4).
14. Delete and recreate a service.
15. Change the parameter of (b) from 1 to 2, (d) from 0 to 100 in (2).
16. Repeat (3) to (4).
17. Delete and recreate a service.
18. Change the parameter of (f) from 3 to 5 in (2).
19. Repeat (3) to (4).
20. Delete and recreate a service.
21. Change the parameter of (e) from 0xFFFFFFFF to 100, (f) from 3 to 0xFF in (2).
22. Repeat (3) to (4).
23. Check the cycle is more than 0xFF times.
24. Verify that after this test scenario, no error occurs.

* 1. Multiple answer Test (netx\_mdns\_multiple\_answer\_test.c). This scenario verifies when there are two services matching the query, both services are present in the response to the query.

Test Procedure:

1. Create mDNS instance.
2. Add two services and the types are the same, “\_http.\_tcp”.
3. Inject a query for type “\_http.\_tcp”.
4. Check whether there are two answers in the response.
5. Verify that after this test scenario, no error occurs.
   1. Responder cooperation Test (netx\_mdns\_responder\_cooperating\_test.c). This scenario verifies four situations at page 21, section 6.6, RFC 6762:
6. While receiving an answer with same name, type, class but different rdata, and the local RR is shared, and then no action is required.
7. While receiving an answer with same name, type, class but different rdata, and the local RR is unique, and then conflict resolution process is required.
8. While receiving an answer with same name, type, class, rdata, and the received TTL is at least half of local one, and then no action is required.
9. While receiving an answer with same name, type, class, rdata, and the received TTL is less than half of local one, and then response is required.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Inject a DNS-SD response that matches situation (a).
4. Check no packets are sent.
5. Inject a response that matches situation (b).
6. Check probing and announcement are performed.
7. Delete the service and add a new service.
8. Inject a response that matches situation (c).
9. Check no packets are sent.
10. Delete the service and add a new service.
11. Inject a response that matches situation (d).
12. Check response is sent.
13. Verify that after this test scenario, no error occurs.
    1. Response with question Test (netx\_mdns\_response\_with\_question\_test.c). This scenario verifies that mDNS module does not process questions in response packet ( Page 14, Section 6, RFC 6762)

Test Procedure:

1. Create mDNS instance.
2. Add a service with type “\_printer.\_tcp”.
3. Inject a response with question for type “\_printer.\_tcp”.
4. Check whether the RR in response is stored.
5. Check whether or not mDNS transmits a response to the question.
6. Verify that after this test scenario, no error occurs.
   1. Source address Test (netx\_mdns\_source\_address\_test.c). This scenario verifies that the response received from a unicast destination address, the source address should be on the local subnet.

Test Procedure:

1. Create mDNS instance.
2. Set the address of first interface of ip\_0 to 1.2.3.4/24. Set the address of second interface of ip\_0 to 2.2.3.4/24. Set the address of first interface of ip\_1 to 1.2.3.5/24.
3. Inject a response from 1.2.3.5 to 2.2.3.4.
4. Check the RR is not stored in peer cache.
5. Inject a response from 1.2.3.5 to 1.2.3.4.
6. Check the RR is stored in peer cache.
7. Verify that after this test scenario, no error occurs.
   1. Source port Test (netx\_mdns\_source\_port\_test.c). This scenario verifies that a response packet sent from UDP port other than 5353 must be ignored. (Page 15, Section 6 RFC 6762).

Test Procedure:

1. Create mDNS instance.
2. Create a UDP socket and bind to port 53. Send a response from this UDP socket.
3. Call service lookup to check whether RR in the response is dropped.
4. Verify that after this test scenario, no error occurs.
   1. Two buffers Test (netx\_mdns\_two\_buffer\_test.c). This scenario verifies that local cache and peer cache are correctly used. Local cache is used by service add. Peer cache is used by query and RR received.

Test Procedure:

1. Create mDNS instance with cache size too small to add a service.
2. Add a service.
3. Check local cache is empty and peer cache is empty.
4. Delete mDNS instance and recreate mDNS instance with cache size 5k.
5. Add a service.
6. Check local cache is not empty and peer cache is empty.
7. Delete the service.
8. Check local cache is empty and peer cache is empty.
9. Add a query.
10. Check local cache is empty and peer cache is not empty.
11. Delete the query.
12. Check local cache is empty and peer cache is empty.
13. Verify that after this test scenario, no error occurs.
    1. Buffer size Test (netx\_mdns\_buffer\_size\_test.c). This scenario verifies that the local or peer cache should not cause a crash no matter how small or large they are.

Test Procedure:

1. Select a cache size from 8 (cache header plus cache tail are 8 bytes) to 1000, in increments of 60.
2. Create mDNS instance with specified cache size.
3. Perform service add, query, service delete, query delete operations.
4. Delete the mDNS instance.
5. Repeat (1) to (4).
6. Verify that after this test scenario, no error occurs.
   1. TTL Test (netx\_mdns\_ttl\_test.c). This scenario verifies that the TTL field in IPv4 header for mDNS packet should be 255. (Page 38, Section 11, RFC 6762).

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Check the TTL field in IPv4 header for mDNS packets.
4. Verify that after this test scenario, no error occurs.
   1. TXT Test (netx\_mdns\_txt\_test.c). This scenario verifies that the max length of valid TXT data is 255. Page 12, Section 6.1, RFC 6763.

Test Procedure:

1. Create mDNS instance.
2. Add a service with TXT length 255.
3. Check the service is added successfully.
4. Add a service with TXT length 256.
5. Check the service is not added.
6. Verify that after this test scenario, no error occurs.
   1. TXT notation Test (netx\_mdns\_txt\_notation\_test.c). This scenario verifies that the TXT records is formatted in a “key=value” notation with ‘;’ acting as separator when more than one key is available.

Test Procedure:

1. Create mDNS instance.
2. Inject a response with TXT “paper=A4” and “version=01”.
3. Get TXT record and check the data is “paper=A4;version=01”.
4. Verify that after this test scenario, no error occurs.
   1. Name Test (netx\_mdns\_name\_test.c). This scenario verifies that the max length of instance name is 63. Because there need to be 4 bytes for conflict resolution. The max length of instance name from the application is limited to 59.

Test Procedure:

1. Create mDNS instance.
2. Add a service with length of instance name 59.
3. Add a query with length of instance name 59.
4. Check whether the return status is success.
5. Add a service with length of instance name 60.
6. Check whether the return status is error.
7. Add a query with length of instance name 60.
8. Check whether the return status is error.
9. Verify that after this test scenario, no error occurs.
   1. Domain name Test (netx\_mdns\_domain\_name\_test.c). This scenario verifies that the default domain is “.local” and user can specify a different value.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Check the domain name in probing and announcement packets. It should be “.local”.
4. Change domain name to “.home.local”.
5. Add a new service.
6. Check the domain name in probing and announcement packets. It should be “.home.local”.
7. Change domain name to “.local”.
8. Add a query.
9. Check the domain name in query packet. It should be “.local”.
10. Change domain name to “home.local”.
11. Add a new query.
12. Check the domain name in query packet. It should be “home.local”.
13. Verify that after this test scenario, no error occurs.
    1. Interface Test (netx\_mdns\_interface\_test.c). This scenario verifies that mDNS packets received from interfaces not enabled in mDNS are ignored.

Test Procedure:

1. Create mDNS instance.
2. Enable the second interface (interface index 1) for mDNS.
3. Inject a response packet from the first interface (interface index 0).
4. Check no RRs are stored in peer cache.
5. Enable the first interface (interface index 0) for mDNS.
6. Inject a response packet from the first interface (interface index 0).
7. Check RRs are stored in peer cache.
8. Verify that after this test scenario, no error occurs.
   1. Second interface Test (netx\_mdns\_second\_interface\_test.c). This scenario verifies that mDNS packets received from interfaces not enabled in mDNS are ignored.

Test Procedure:

1. Create mDNS instance.
2. Enable the second interface (interface index 1) for mDNS .
3. Register one service.
4. Check if the interface of probing and announcement messages is the second interface (interface index 1).
5. Start mDNS query.
6. Check if the interface of query is the second interface (interface index 1).
7. Verify that after this test scenario, no error occurs.
   1. Peer service change notify Test (netx\_mdns\_peer\_service\_change\_notify\_test.c). This scenario verifies that the service change notify function can be correctly called when new service received.

Test Procedure:

1. Create mDNS instance.
2. Enable mDNS.
3. Inject new service.
4. Check if the service change notify is called.
5. Lookup service to check if new service is stored.
6. Clear the peer cache.
7. Set the service change notify function for specified service mask.
8. Reinject the new service.
9. Check if the service change notify is called.
10. Verify that after this test scenario, no error occurs.
    1. IPv6 string Test (netx\_mdns\_ipv6\_string\_test.c). This scenario verifies that the string can be correctly stored in cache.

Test Procedure:

1. Create mDNS instance.
2. Enable mDNS.
3. Inject service response to store PTR, AAAA, SRV, TXT resource records into cache.
4. Inject service response to store dns-sd PTR record into cache.
5. Check resource record count and resource record value.
6. Check the string tail pointer.
7. Delete dns-sd PTR record from cache.
8. Recheck the string tail pointer.
9. Verify that after this test scenario, no error occurs.
   1. Address change Test (mdns\_address\_change\_test.c). This scenario verifies when address of the mDNS node is changed, the address record needs to be announced again. (Page 31, Section 8.4, RFC 6762).

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Check the probing and announcement packets.
4. Change the Ipv4 address.
5. Check the announcement packets.
6. Verify that after this test scenario, no error occurs.
   1. Case insensitivity Test (mdns\_case\_insensitivity\_test.c). This scenario verifies that the name of RR name is case-insensitive. (Page 46, Section 16, RFC 6762.)

Test Procedure:

1. Create mDNS instance with host name “ARMMDNSTest”.
2. Inject a query with name “ARMMDNSTEST” type A.
3. Verify that a response is transmitted.
4. Verify that after this test scenario, no error occurs.
   1. Query during probing Test (mdns\_query\_during\_probing\_test.c). This scenario verifies that a service does not answer a query when it is in probing state.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Check the first probing packet.
4. Inject a query for that service.
5. Verify that no responses are observed.
6. Wait till announcement finish.
7. Inject a query for that service.
8. Verify a response is transmitted.
9. Verify that after this test scenario, no error occurs.
   1. Server interface reset Test (mdns\_server\_interface\_reset\_test.c). This scenario verifies when an interface is disabled and then enabled (effective a reset), mDNS should perform probing and announcement again on the registered RRs that are unique.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Check the probing and announcement packets.
4. Disable the interface.
5. Check goodbye packet is sent.
6. Enable the interface.
7. Check the probing and announcement packets.
8. Verify that after this test scenario, no error occurs.
   1. Server send goodbye Test (mdns\_server\_send\_goodbye\_test.c). This scenario verifies before a service is deleted, goodbye response is sent with TTL 0.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Delete the service
4. Verify that goodbye packet is transmitted.
5. Verify that after this test scenario, no error occurs.
   1. DNS-SD response Test (mdns\_dns\_sd\_response\_test.c). This scenario verifies that mDNS module can reply to “\_services.\_dns-sd.\_udp” query.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Inject the DNS-SD query packet.
4. Check the response to DNS-SD query.
5. Verify that after this test scenario, no error occurs.
   1. Probing conflict Test (mdns\_probing\_conflict\_test.c). This scenario verifies when there’s conflict during probing, the name should be changed automatically.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Check the first probing packet.
4. Inject a response to answer the probing packet indicating a conflict.
5. Check the probing packet with name changed.
6. Check the service\_change\_notify callback. It should be invoked with state NX\_SERVICE\_NAME\_REGISTERED.
7. Verify that after this test scenario, no error occurs.
   1. Server announcement with TXT Test (mdns\_server\_announcement\_with\_txt\_test.c). This scenario verifies that the interval between probing and announcement packets obeys parameters specified in RFC6762. The delay of first probing is 0-250ms. The delay of following probing packets is 250ms. The delay of probing and announcement packets is 250ms. The delay of following announcement packets is increased by exponentially.

Test Procedure:

1. Create mDNS instance.
2. Add a service with TXT non-null.
3. Check the first probing packet.
4. Check the interval from (2) to (3) should be 0~260ms (10ms for tolerance).
5. Check the second probing packet.
6. Check the interval from (3) to (5) should be 0~260ms (10ms for tolerance).
7. Check the third probing packet.
8. Check the interval from (5) to (7) should be 0~260ms (10ms for tolerance).
9. Check the first announcement packet.
10. Check the interval from (7) to (9) should be 0~260ms (10ms for tolerance).
11. Check the second announcement packet.
12. Check the interval from (9) to (11) should be 1s±30ms (30ms for tolerance).
13. Check the second announcement packet.
14. Check the interval from (11) to (13) should be 2s±30ms (30ms for tolerance).
15. Verify that after this test scenario, no error occurs.
    1. Response no delay Test (mdns\_response\_no\_delay\_test.c). This scenario verifies that if the responder ensures it is the only responder that will send the response, it should be sent within 10ms. Page 14, Section 6, RFC 6762.

Test Procedure:

1. Create mDNS instance with host name “ARMMDNSTest”.
2. Inject a query with name “ARMMDNSTEST” type A.
3. Check the response packet.
4. Check the interval from (2) to (3) should be 0~10ms.
5. Verify that after this test scenario, no error occurs.
   1. Response interval Test (mdns\_response\_interval\_test.c). This scenario verifies that in normal case, response should be sent during 20~120ms. A response should not be sent twice when there’re two queries for this RR.

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Inject a query type PTR.
4. Check the response packet.
5. Check the interval from (3) to (4) should be 20~120ms.
6. Wait 5 seconds and repeat (3) to (5).
7. Inject a query for type PTR immediately.
8. Check the response packet.
9. Check the interval from (7) to (8) should be more than 1s.
10. Wait 5 seconds and repeat (3) to (5).
11. Inject a query type SRV.
12. Check the response packet.
13. Check the interval from (9) to (10) should be 0~10ms.
14. Inject a probing for this service immediately.
15. Check the response packet.
16. Check the interval from (10) to (13) should be more than 250ms.
17. Verify that after this test scenario, no error occurs.
    1. Response aggregation Test (mdns\_response\_aggregation\_test.c). This scenario verifies that when there’re two or more responses to be transmitted, they must be sent in one packet.( Page 20, Section 6.4, RFC 6762)

Test Procedure:

1. Create mDNS instance.
2. Add two services.
3. Inject the first query.
4. Sleep 10ms and inject the second query.
5. Check there’s only one response packet and it contains answers to both queries.
6. Verify that after this test scenario, no error occurs.
   1. Response with TC Test (mdns\_response\_with\_tc\_test.c). This scenario verifies that when Responders received one query with TC (truncated) bit set, Responders SHOULD delay their responses by a random amount of time selected with uniform random distribution in the range 400-500ms. (Page 23, Section 7.2, RFC 6762.). In order to run this test case, the following symbols must be defined

Compilation Options: #define NX\_MDNS\_ENABLE\_NEGATIVE\_RESPONSES

Test Procedure:

1. Create an mDNS instance and initialize the cache with the default cache size.
2. Add a service.
3. Wait for service probing and announcing to finish
4. Inject a query.
5. Verify the response packet.
6. Inject a query with TC bit set.
7. Verify the response packet, to be in the range of 400 -500ms.
8. Inject a query with known answer.
9. Verify no responses are transmitted.
10. Verify that after this test scenario, no error occurs.
    1. Known answer suppression response Test (mdns\_known\_answer\_suppression\_response.c). This scenario verifies while receiving a query with known answer, the RRs that match the known answers should not be included the response. When the TTL of known answer is less than half of the TTL of the local RR, the reply should be included in the response. (Page 23, Section 7.1, RFC 6762.)

Test Procedure:

1. Create mDNS instance.
2. Add a service.
3. Inject query packet with known answer.
4. Verify no responses are transmitted.
5. Inject query packet without known answer section.
6. Check response to the query.
7. Inject query packet with known answer.
8. Verify no responses are replied.
9. Inject a query packet with known answer. The TTL of known answer is less than half of the value in the local RR.
10. Verify the responses to the query.
11. Verify that after this test scenario, no error occurs.
    1. Known answer suppression unique Test (mdns\_known\_answer\_suppression\_unique\_test.c). This scenario verifies while there is a unique answer to a query, the query should not be sent. (Page 23, Section 7.1, RFC 6762.)

Test Procedure:

1. Create mDNS instance.
2. Add a service with name “ARMMDNSTest. \_ipp.\_tcp”.
3. Add a query to “ARMMDNSTest. \_ipp.\_tcp”.
4. Since there is a unique answer in local cache, no query packet is sent.
5. Verify that after this test scenario, no error occurs.
   1. Duplicate answer suppression Test (mdns\_duplicate\_answer\_suppression\_test.c). This scenario verifies that a responder should not send a response when it receives same answer with TTL not less than the one of responder would send. Page 24, Section 7.4, RFC 6762.

Test Procedure:

1. Create mDNS instance.
2. Add a service with TTL 4500.
3. Inject a query packet for that service.
4. Inject a response with the same answer created in (2). The TTL of the response is 4500.
5. Verify that no responses are transmitted.
6. Inject a query packet for that service.
7. Inject a response with the same answer created in (2). The TTL of the response is 4499.
8. Verify that a response is transmitted.
9. Verify that after this test scenario, no error occurs.
   1. Announcement in multiple packets Test (mdns\_announcement\_in\_multiple\_packets\_test.c). This scenario verifies that when a mDNS Responder is able to divide multiple RRs into multiple packets if they don’t fit into one MTU.

In order to run this test case, the following symbols must be defined

Compilation Options: #define NX\_MDNS\_ENABLE\_NEGATIVE\_RESPONSES

Test Procedure:

1. Create an mDNS instance and initialize the cache with the default cache size.
2. Add fourteen services.
3. Verify six probing packets are transmitted, the packets includes the all probing records.
4. Verify nine announcing packets are transmitted, the packets includes the all announcing records.
5. Verify that after this test scenario, no error occurs.
   1. Response in multiple packets Test (mdns\_response\_in\_multiple\_packets\_test.c). This scenario verifies that when a MDNS Responder has too many answers to fit in MTU, its divides the answer list into two or more packets.

In order to run this test case, the following symbols must be defined

Compilation Options: #define NX\_MDNS\_ENABLE\_NEGATIVE\_RESPONSES

Test Procedure:

1. Create an mDNS instance and initialize the cache with the default cache size.
2. Add fourteen services.
3. Wait for service probing and announcing to finish.
4. Inject the query.
5. Verify three responses are transmitted, the responses includes the all answer and additional answer.
6. Verify that after this test scenario, no error occurs.
   1. Basic IPv6 response Test (mdns\_basic\_ipv6\_response\_test.c). This scenario verifies that mDNS module responder can process query and send response over IPv6 correctly.

Test Procedure:

1. Add IPv6 address and create mDNS instance.
2. Add a service.
3. Inject a query packet over IPv6.
4. Check the response over IPv6.
5. Inject a query packet with known answer over IPv6.
6. Check no response packet over IPv6 is sent.
7. Verify that after this test scenario, no error occurs.
   1. Basic IPv6 announcement Test (mdns\_basic\_ipv6\_announcement\_test.c). This scenario verifies that mDNS module can send probing and announcement packets over IPv6.

Test Procedure:

1. Add IPv6 address and create mDNS instance.
2. Add a service.
3. Verify three probing packets over IPv6.
4. Verify three announcement packets over IPv6.
5. Verify that after this test scenario, no error occurs.
   1. Response to address query Test (mdns\_response\_to\_address\_query\_test.c). This scenario verifies when mDNS responder sends one of its addresses then it should send all of its addresses. (Page 19, Section 6.2, RFC 6762.)

Test Procedure:

1. Add IPv6 address and create mDNS instance.
2. Inject a query for type A.
3. Verify the response packet contains A and AAAA RRs.
4. Verify that after this test scenario, no error occurs.
   1. Passive observation of failures (POOF)Test (mdns\_poof\_test.c). This scenario verifies that when there are two or more queries but no response in ten seconds, the information stored in the peer service cache shall be considered invalid. (Page 38, Section 10.5, RFC 6762.)

Test Procedure:

1. Create mDNS instance.
2. Inject a response.
3. Check the response is stored in peer cache.
4. Inject two queries for that response.
5. Wait for eleven seconds (more than ten seconds)
6. Check that the RR answers the query is removed.
7. Verify that after this test scenario, no error occurs.
   1. DNS-SD query Test (mdns\_dns\_sd\_query\_test.c). This scenario verifies that mDNS module can send “\_services.\_dns-sd.\_udp” query to list all service types.

Test Procedure:

1. Create mDNS instance.
2. Add a query with the name and type set to NULL.
3. Verify the DNS-SD query packets.
4. Inject four DNS-SD responses which contain ten RR.
5. Check the count of RR in peer cache. It should be ten.
6. Verify that after this test scenario, no error occurs.
   1. Known answer ignored Test (mdns\_know\_answer\_ingored\_test.c). This scenario verifies that RR in known answer section should not be cached. Page 23, Section 7.1, RFC 6762.

Test Procedure:

1. Create mDNS instance.
2. Inject a query with known answer.
3. Check there’s only one response packet and it contains both answers.
4. Verify that after this test scenario, no error occurs.
   1. Known answer suppression query Test (mdns\_known\_answer\_suppression\_query\_test.c). This scenario verifies while sending a query, it should add all known answers to this query. (Page 22, Section 7.1, RFC 6762).

Test Procedure:

1. Create mDNS instance.
2. Add a query.
3. Check the query packet.
4. Inject a response to the query.
5. Verify the query packet sent that contains known answer section.
6. Delete the query and add it again.
7. Verify the query packet sent that contains known answer section.
8. Verify that after this test scenario, no error occurs.
   1. Known answer suppression query half TTL Test (mdns\_known\_answer\_suppression\_query\_half\_ttl\_test.c). This scenario verifies while half TTL of a RR has past, the RR should not be in known answer section. (Page 23, Section 7.1, RFC 6762).

Test Procedure:

1. Create mDNS instance.
2. Add a query.
3. Check the query packet.
4. Inject a response to the query.
5. Wait half TTL and check the query packet without known answer for half TTL seconds.
6. Verify that after this test scenario, no error occurs.
   1. Duplicate question suppression Test (mdns\_duplicate\_question\_suppression\_test.c). This scenario verifies that during continuous query, if it receives the same query from the network, it should treat it as if this query has been sent. But if there are known answers in our query or their query, our query is transmitted anyway. (Page 24, Section 7.3, RFC 6762.)

Test Procedure:

1. Create mDNS instance.
2. Add a query.
3. Check the query packet.
4. Inject a query which is the same as (3).
5. Check no query is sent in one second (the interval between first and second query is one second).
6. Inject a query which is the same as (3) but with QU bit set. It doesn't affect the query interval.
7. Inject a query with known answer. It doesn’t affect the query interval.
8. Check the query packet in two seconds.
9. Inject a response packet.
10. Check the query packet with known answer in 4 seconds.
11. Inject a query which is same as (3). The interval is not affect since local query contains known answer.
12. Check the query packet with known answer in 8 seconds.
13. Verify that after this test scenario, no error occurs.
    1. Continuous query Test (mdns\_continuous\_query\_test.c). This scenario verifies that during continuous query, the known answer section is contains answers from responses received. And queries are sent continuously.

Test Procedure:

1. Create mDNS instance.
2. Add a query.
3. Verify the query packet is transmitted.
4. Inject a response packet.
5. Check the query packet is sent with known answer section.
6. Repeat four times of (5).
7. Verify that after this test scenario, no error occurs.
   1. Continuous query unique Test (mdns\_continuous\_query\_unique\_answer\_test.c). This scenario verifies that during continuous query, it can be stopped when unique answer is received. (Page 11, Section 5.2, RFC 6762.)

Test Procedure:

1. Create mDNS instance.
2. Add a query with name and type specified.
3. Check the query packet.
4. Inject a response with unique answer (cache-flush bit is set).
5. Verify that no more responses are sent.
6. Verify that after this test scenario, no error occurs.
   1. Continuous query interval Test (mdns\_continuous\_query\_interval\_test.c). This scenario verifies that the delay of first query is 20~120ms. The interval of first two queries is one second. And the interval of successive queries is increased by a factor of two. (Page 9, Section 5.2, RFC 6762.)

Test Procedure:

1. Create mDNS instance.
2. Add a query.
3. Check the first query packet.
4. Check the interval is in range 20~120ms.
5. Check the second query packet.
6. Check the interval is 1s±30ms.
7. Check the third query packet.
8. Check the interval is 2s±30ms.
9. Check the fourth query packet.
10. Check the interval is 4s±30ms.
11. Check the fifth query packet.
12. Check the interval is 8s±30ms.
13. Verify that after this test scenario, no error occurs.
    1. Query start and stop Test (mdns\_query\_start\_stop\_test.c). This scenario verifies that query can be start and stop correctly.

Test Procedure:

1. Create mDNS instance.
2. Add a query for “test.\_http.\_tcp”. The sub type is set to “\_printer” which should be ignored since service name does not contain sub type.
3. Check the query packet is sent and delete this query.
4. Check no more query packets are sent.
5. Add a query for “test.\_http.\_tcp”.
6. Check the query packet is sent.
7. Inject a response packet.
8. Check no more query packets are sent since a unique answer is received.
9. Delete the query.
10. Add a query for “\_printer.\_sub.\_http.\_tcp”.
11. Check the query packet is sent and delete this query.
12. Check no more query packets are sent.
13. Add a query for “\_printer.\_sub.\_http.\_tcp”.
14. Check the query packet is sent.
15. Inject a response packet.
16. Check the query packet is sent and known answer section is filled.
17. Delete the query.
18. Check no more query packets are sent.
19. Verify that after this test scenario, no error occurs.
    1. Query http Test (mdns\_query\_http\_tcp\_test.c). This scenario verifies that client can query http protocol.

Test Procedure:

1. Create mDNS instance.
2. Add a query for “\_http.\_tcp”.
3. Verify the query packet.
4. Inject a response.
5. Verify the query packet contains known answer from the response packet.
6. Verify that the service\_change\_notify function is called.
7. Verify that after this test scenario, no error occurs.
   1. Query pdl-datastream Test (mdns\_query\_pdl\_datastream\_tcp\_test.c). This scenario verifies that client can query pdl-datastream protocol.

Test Procedure:

1. Create mDNS instance.
2. Add a query for “\_pdl-datastream.\_tcp”.
3. Verify the query packet.
4. Inject a response.
5. Verify the query packet contains known answer from the response packet.
6. Verify that after this test scenario, no error occurs.
   1. Query printer Test (mdns\_query\_printer\_tcp\_test.c). This scenario verifies that client can query printer protocol.

Test Procedure:

1. Create mDNS instance.
2. Add a query for “\_printer.\_tcp”.
3. Verify the query packet.
4. Inject a response.
5. Verify the query packet contains known answer from the response packet.
6. Verify that after this test scenario, no error occurs.
   1. Query smb Test (mdns\_query\_smb\_tcp\_test.c). This scenario verifies that client can query smb protocol.

Test Procedure:

1. Create mDNS instance.
2. Add a query for “\_smb.\_tcp”.
3. Verify the query packet.
4. Inject a response.
5. Verify the query packet contains known answer from the response packet.
6. Verify that after this test scenario, no error occurs.
   1. Query and response chaos Test (mdns\_query\_and\_response\_chaos\_test.c). This scenario verifies that mDNS module is not affected by chaos packets from network.

Test Procedure:

1. Create mDNS instance.
2. Inject ten packets containing queries and responses. There are 5 RRs in these packets.
3. Check the received RR number is 5.
4. Verify that after this test scenario, no error occurs.
   1. Query with TC Test (mdns\_query\_with\_tc\_test.c). This scenario verifies that when a MDNS querier will already have too many answers to fit in the Known-Answer Section of its query packet. It MUST then set the TC (truncated) bit in the header before sending the query. (Page 23, Section 7.2, RFC 6762.)

Test Procedure:

1. Create an mDNS instance and initialize the cache with the default cache size.
2. Add a query.
3. Inject response with 14 answer records for the query.
4. Inject response with 2 answer records for the query.
5. Verify the one query packet contains TC bit and include 1 query record and 15 known answer record.
6. Verify the other query packet contains 1 known answer.
7. Verify that after this test scenario, no error occurs.
   1. Query RR timeout Test (mdns\_query\_rr\_timeout\_test.c). This scenario verifies that a continuous querier should send its query at 80%, 85%, 90% and 95% of TTL if no answer is received. And the RR is deleted when 100% of its TTL reaches. When there’s no query on a RR, it is deleted at 100% of its TTL directly.

Test Procedure:

1. Create mDNS instance.
2. Add a query.
3. Verify the query packet.
4. Inject a response packet with TTL 120.
5. Check the query packet at 96 (80%) seconds.
6. Check the query packet at 102 (85%) seconds.
7. Check the query packet at 108 (90%) seconds.
8. Check the query packet at 114 (95%) seconds.
9. Check the RR count in peer cache.
10. Check the response should have been deleted at 120 (100%) seconds.
11. Delete the query of (2).
12. Inject a response packet with TTL 120.
13. Check no query is sent in 120 (100%) seconds.
14. Check the response should have been deleted.
15. Verify that after this test scenario, no error occurs.
    1. Multiple questions per query Test (mdns\_multiple\_questions\_per\_query\_test.c). This scenario verifies when there are multiple questions in a short period of time (120ms in our mDNS), these queries (questions) are sent in one query packet.

Test Procedure:

1. Create mDNS instance.
2. Add three queries.
3. Verify that the query contains three questions.
4. Verify that after this test scenario, no error occurs.
   1. Client passive Test (mdns\_client\_passive\_test.c). This scenario verifies that in passive mode, responses received from the network are processed. While receiving goodbye packets with TTL 0, RR is not deleted immediately. It is deleted after one second.

Test Procedure:

1. Create mDNS instance.
2. Inject three probing packets.
3. Verify that no RR is stored.
4. Inject an announcement packet with 6 RRs.
5. Verify that all 6 RRs are stored in peer cache. Verify that the service is correct.
6. Inject another two announcement packets same as (4).
7. Verify there are 6 RRs in peer cache. Verify that the service is correct.
8. Inject a goodbye packet indicates 4 RRs are leaving.
9. Verify that there are 6 RRs in peer cache immediately.
10. Wait one second and check there are 2 RRs in peer cache.
11. Inject a response packet with 1 RR.
12. Verify there’re 3 RRs in peer cache.
13. Verify that after this test scenario, no error occurs.
    1. Basic IPv6 query Test (mdns\_basic\_ipv6\_query\_test.c). This scenario verifies that mDNS module querier can send query and process response over IPv6 correctly.

Test Procedure:

1. Add IPv6 address and create mDNS instance.
2. Add a query.
3. Verify that the query packet is transmitted over IPv6.
4. Inject a response over IPv6.
5. Verify that the query packet contains known answer from the response and is transmitted over IPv6.
6. Verify that after this test scenario, no error occurs.
7. Test Report

Copyright (c) 1996-2013 Express Logic Inc. \* NetX Duo Cortex-M4/CCS Version G5.7.5.2 SN: 3382-268-0715 \*

IPv6 is built-in.

IPsec is not built-in.

Tunnel is not built-in.

IP structure size: 4164

TCP control block size: 312

ARP table entry size: 52, ARP table size 1664

Packet structure size: 60

NetX Test: MDNS Internal Function Test...............................SUCCESS!

NetX Test: MDNS Create Delete Test...................................SUCCESS!

NetX Test: MDNS One Shot Query Test..................................SUCCESS!

NetX Test: MDNS Local Cache Continuous Query Test....................SUCCESS!

NetX Test: MDNS Local Cache One Shot Query Test......................SUCCESS!

NetX Test: MDNS Service Lookup Test..................................SUCCESS!

NetX Test: MDNS Service Add And Delete Test..........................SUCCESS!

NetX Test: MDNS Announcement Repeat Test.............................SUCCESS!

NetX Test: MDNS Multiple Answers Test................................SUCCESS!

NetX Test: MDNS Responder Cooperating Test...........................SUCCESS!

NetX Test: MDNS Response With Question Test..........................SUCCESS!

NetX Test: MDNS Source Address Test..................................SUCCESS!

NetX Test: MDNS Source Port Test.....................................SUCCESS!

NetX Test: MDNS Two Buffer Test......................................SUCCESS!

NetX Test: MDNS Buffer Size Test.....................................SUCCESS!

NetX Test: MDNS TTL Test.............................................SUCCESS!

NetX Test: MDNS TXT Test.............................................SUCCESS!

NetX Test: MDNS TXT Notation Test....................................SUCCESS!

NetX Test: MDNS Name Test............................................SUCCESS!

NetX Test: MDNS Domain Name Test.....................................SUCCESS!

NetX Test: MDNS Interface Test.......................................SUCCESS!

NetX Test: MDNS Second Interface Test................................SUCCESS!

NetX Test: MDNS Peer Service Change Notify Test......................SUCCESS!

NetX Test: MDNS IPv6 String Test.....................................SUCCESS!

NetX Test: MDNS Address change TEST..................................SUCCESS!

NetX Test: MDNS Case-insensitivity TEST..............................SUCCESS!

NetX Test: MDNS Query during probing TEST............................SUCCESS!

NetX Test: MDNS Server interface reset TEST..........................SUCCESS!

NetX Test: MDNS Server send goodbye TEST.............................SUCCESS!

NetX Test: MDNS DNS-SD response TEST.................................SUCCESS!

NetX Test: MDNS Probing conflict TEST................................SUCCESS!

NetX Test: MDNS Server announcement with TXT TEST....................SUCCESS!

NetX Test: MDNS Response no delay TEST...............................SUCCESS!

NetX Test: MDNS Response interval TEST...............................SUCCESS!

NetX Test: MDNS Response aggregation TEST............................SUCCESS!

NetX Test: MDNS Response in multiple packets TEST....................SUCCESS!

NetX Test: MDNS Response with TC TEST................................SUCCESS!

NetX Test: MDNS Known answer suppression response TEST...............SUCCESS!

NetX Test: MDNS Known answer suppression unique TEST.................SUCCESS!

NetX Test: MDNS Duplicate answer suppression TEST....................SUCCESS!

NetX Test: MDNS Announcement in multiple packets TEST................SUCCESS!

NetX Test: MDNS Basic IPv6 response TEST.............................SUCCESS!

NetX Test: MDNS Basic IPv6 announcement TEST.........................SUCCESS!

NetX Test: MDNS Response to address query TEST.......................SUCCESS!

NetX Test: MDNS Passive observation of failures TEST.................SUCCESS!

NetX Test: MDNS DNS-SD query TEST....................................SUCCESS!

NetX Test: MDNS Known answer ignored TEST............................SUCCESS!

NetX Test: MDNS Known answer suppression query TEST..................SUCCESS!

NetX Test: MDNS Known answer suppression query half TTL TEST.........SUCCESS!

NetX Test: MDNS Duplicate question suppression TEST..................SUCCESS!

NetX Test: MDNS Continuous query TEST................................SUCCESS!

NetX Test: MDNS Continuous query unique answer TEST..................SUCCESS!

NetX Test: MDNS Continuous query interval TEST.......................SUCCESS!

NetX Test: MDNS Query start and stop TEST............................SUCCESS!

NetX Test: MDNS Query \_http.\_tcp TEST................................SUCCESS!

NetX Test: MDNS Query \_pdl-datastream.\_tcp TEST......................SUCCESS!

NetX Test: MDNS Query \_printer.\_tcp TEST.............................SUCCESS!

NetX Test: MDNS Query \_smb.\_tcp TEST.................................SUCCESS!

NetX Test: MDNS Query and response TEST..............................SUCCESS!

NetX Test: MDNS Query with TC TEST...................................SUCCESS!

NetX Test: MDNS RR timeout TEST......................................SUCCESS!

NetX Test: MDNS Multiple questions per query TEST....................SUCCESS!

NetX Test: MDNS Client passive TEST..................................SUCCESS!

NetX Test: MDNS Basic IPv6 query TEST................................SUCCESS!

\*\*\*\* Testing Complete \*\*\*\*

\*\*\*\* Test Summary: Tests Passed: 64 Tests Warning: 0 Tests Failed: 0