1. Test Setup

NetX Validation System is a platform that executes Test Scenarios. Each Test Scenario is a NetX application that targets a specific functionality or a group of related functionalities, and a peer application that interacts with the test application. To start a scenario, NetX Validation System creates two IP instances. IP instance 0 (ip\_0) is typically used to run the test application. IP instance 1 (ip\_1) typically serves the peer application. The two IP instances are both connected at the RAM-based simulated Ethernet device driver (the RAM driver). To test a mutli-home system, multiple RAM drivers can be attached to an IP instance. Data packets transmitted by the application traverse the entire NetX stack down to the RAM driver. The RAM driver dispatches the packet to the appropriate interface of the destination IP instance, simulating a typical Ethernet network environment. Incoming packets are received by the RAM driver and are forwarded to the IP instance. NetX validation system is able to create various network situations, such as simulating a packet drop, an out-of-order packet, or a mal-formatted packet. Between Test Scenarios, NetX Validation system cleans up all objects and internal data structures. Therefore the result from one Test Scenario does not affect the performance of the next test. As the Test Scenario runs, the name of the test, and the test results are printed. At the end of the test, a summary of the test (total number of tests executed, and the number of tests pass) is displayed.

1. Test Scenarios

This section describes NetX Test Scenarios.

* 1. TCP Disconnect Test. This scenario verifies that the BSD layer is able to handle rapid connect/disconnect sequence in the IPv4 network without losing pending connections, and would not have internal data structure corruptions.

Test Procedure:

1. Initialize ip\_0 and configure one IPv4 address.
2. Test a single TCP Client connection over the IPv4 network. Once connected, the client sends a message and closes down.
3. Repeat step (2) for 3000 times.
4. Create a TCP server thread.
5. Create four TCP client threads.
6. The client threads runs concurrently, all attempt to connect to the same server.
7. Upon a client connection, the server creates a helper thread to handle the connection.
8. Each helper thread receives a short message from the client and disconnects, and terminates.
9. Repeat (6)-(8) so the server socket accepts total 6000 connections.
10. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
    1. TCP Multiple Accept Test. This scenario tests two concurrent TCP servers accepting connections on two different port numbers from 2 clients.

Test Procedure:

1. Initialize ip\_0 and configure one IPv4 address.
2. Create 2 TCP server threads, each binding to a different port.
3. Upon a TCP connection, each server would create a helper thread to handle the client connection.
4. Create TCP client sockets to connect to each of the servers.
5. Repeat (4) four times
6. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
   1. Basic BSD TCP Blocking Test. This scenario tests the behavior of BSD TCP sockets over both IPv4 and IPv6 network, as server sockets, and as client sockets. In this test scenario, all BSD sockets are created as blocking socket. For each TCP connection, a new helper thread is created to handle the connection. Over each connection, the socket first receives a simulated request message followed by sending a response message. Both ends verify the connection with the expected peer, the content of each message passed.

Test Procedure:

1. Initialize ip\_0 and configure one IPv4 address and one IPv6 link local address, one IPv6 global address.
2. Test a single TCP Client connection over the IPv4 network. Once connected, the client sends a message and closes down.
3. Test a single TCP server for the IPv4 network, and accepts multiple TCP client connections (10 clients). For each connection, the server thread creates a helper thread to handle the connection. Within the helper thread, the server application receives a message from the client, and sends a response to the client before closing down the socket, and terminates the helper thread.
4. Test a single TCP client connection over the IPv6 network. Once connected, the client sends a message to the server and closes down. This test is similar to (2) except that the connection is made over the IPv6 network.
5. Test a single TCP server for the IPv6 network, and accepts multiple TCP client connections (10 clients). For each connection, the server thread creates a helper thread to handle the connection. Within the helper thread, the server application receives a message from the client, and sends a response to the client before closing down the socket, and terminates the helper thread. This test is similar to (3) except that the connections are made over the IPv6 network.
6. Create a socket and attempt to connect via the IPv4 network to a port that has no server listening on. Verify the connection would fail.
7. Create a socket and attempt to connect via the IPv4 network to an invalid host IP address. Verify the connection would fail.
8. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
   1. Basic BSD UDP Connect Test. This scenario verifies the behavior of the BSD UDP sockets over both IPv4 and IPv6 network. Each UDP socket is connected to a remote host. This scenario verifies that a connected socket is able to send data to the designated peer using the “send” call, and is able to receive only from the designated peer. This scenario also verifies that the socket is able to receive data using the “recv” call. In this test scenario, all BSD sockets are created as blocking socket.

Test Procedure:

1. Initialize ip\_0 with 3 physical interfaces. Each interface is configured with one IPv4 address, one IPv6 link local address, and 2 IPv6 global addresses.
2. Create a UDP socket, bind to a local port, and “connect” to a UDP server on its IPv4 address. Then the UDP socket receives a message using “recv”, and then transmits a response message back to the sender using the “send” call. Verify the other side is able to receive the response message.
3. Create a UDP socket, bind to a local port, and “connect” to a UDP server on its IPv6 address. Then the UDP socket receives a message using “recv”, and then transmits a response message back to the sender using the “send” call. Verify the other side is able to receive the response message.
4. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
   1. Basic BSD TCP Bind Test. This scenario verifies that the behavior of the BSD TCP sockets with various bind options over IPv4 and IPv6 network. No two sockets are allowed to bind to the same TCP port and local interface. When multiple sockets are bind to the same TCP port, they must bind to different local interfaces, or one of them can be bound to INADDR\_ANY. An incoming TCP connection is connected to the TCP socket with the best matching binding address, followed by the socket that is bound to INADDR\_ANY. All the TCP sockets are created as blocking sockets.

Test Procedure:

1. Initialize ip\_0 with 3 physical interfaces. Each interface is configured with one IPv4 address, one IPv6 link local address, and 2 IPv6 global addresses.
2. Create 5 TCP server threads:

Server0: binds to IPv4 local interface 0

Server1: binds to IPv4 local interface 1

Server2: binds to IPv4 INADDR ANY.

Server3: binds to IPv6 global address on interface 1

Server4: Binds to IPv6 any address.

Each server would create a helper thread to handle a connection.

1. Create 2 TCP server (IPv6) threads, one binds to IPv6 global address on the 2nd interface, and one binds to IPv6 any address.
2. Create a TCP client to connect to everyIPv4 addresses and every IPv6 global addresses. Upon a connection, the client sends a message, and waits for the server to send a response.
3. Verify that each TCP connection is routed to the server that is bound to the specific IP local interface. If a local interface does not have a server bound to it, the server that is bound to INADDR\_ANY handles the connection.
4. Terminate the server that binds to IPv4 interface 1 (server1)
5. Rerun test (5), verify that the all TCP client connections are handled.
6. Terminate the server that binds to IPv4 INADDR\_ANY (server2)
7. Rerun test (5). Verify that only IPv4 connections to server0 are handled. All other IPv4 connections should fail. All IPv6 connections should be handled.
8. Terminate the server that binds to IPv4 any address (server4), and create server 5: binds to IPv4 INADDR\_ANY.
9. Rerun (5). Verify that all IPv4 connections should be handled. Only IPv6 connection to server 3 should be handled.
10. Create server5: binds to IPv4 INADDR\_ANY
11. Rerun test (5). Verify that all IPv4 connections should be handled. Only IPv6 connection to server.
12. Shutdown all clients and servers.
13. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
    1. Basic BSD UDP Bind Test. This scenario verifies that the behavior of the BSD UDP sockets with various bind options over IPv4 and IPv6 network. No two sockets are allowed to bind to the same UDP port and local interface. When multiple sockets are bind to the same UDP port, they must bind to different local interfaces, or one of them can be bound to INADDR\_ANY. An incoming UDP packet is delivered to the UDP socket that has the most matching local address information, or a socket that is bound to INADDR\_ANY. All the UDP sockets are created as blocking sockets.

Test Procedure:

1. Initialize ip\_0 with 3 physical interfaces. Each interface is configured with one IPv4 address, one IPv6 link local address, and 2 IPv6 global addresses.
2. Create a UDP server on the IPv4 network and bind to local INADDR\_ANY
3. Send UDP messages to each of the IPv4 addresses, and wait for response from the server. Verify that all UDP servers are responsive.
4. Create a UDP server on the IPv6 network and bind to local INADDR\_ANY
5. Send UDP messages to each of the IPv6 global addresses, and wait for response from the server. Verify that all UDP servers are responsive.
6. Create a UDP server that is bound to IPv4 any address. Verify that it does not receive data on the IPv6 network.
7. Create a UDP server that is bound to IPv6 any address. Verify that it does not receive data on the IPv4 network.
8. Create UDP servers that are bound to each IPv4 interfaces. Verify that they don’t receive data destined for another IPv4 address, or IPv6 address. (this test verifies on UDP server at a time).
9. Create UDP servers that are bound to each IPv6 global address. Verify that they don’t receive data destined for another IPv6 address, or IPv4 address. (this test verifies on UDP server at a time).
10. Create 3 concurrent UDP servers , each binds to one IPv4 local interface address. Verify that each server only receives data destined to its IP address.
11. Create 3 concurrent UDP servers binding to interface0, interface2, and INADDR\_ANY. Verify that each server is able to receive data destined to its local address, and packet sent to interface 1 is handled by server binding to INADDR\_ANY
12. Create 3 concurrent UDP servers binding to 3 IPv6 addresses. Verify that each server only receives data destined for its IPv6 address.
13. Create 3 concurrent UDP servers binding to interfacd0, interface1 IPv6 global address, and IPv6 any address. Verify that the UDP server binding to the any address is able to handle UDP data not destined to interface 0 and interface 1 global addresses.
14. Create a UDP socket binding to the 2nd interface (IPv4 address). Allow the client to send 10 messages prior to the UDP server start receiving. Verify that the UDP server is able to receive all 10 messages.
15. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
    1. Basic BSD Multicast Test. This scenario verifies the BSD setsockopt to join and drop an IPv4 multicast group. All sockets are created as blocking sockets.

Test Procedure:

1. Initialize ip\_0 with one physical interface.
2. Create a UDP socket, use setsockopt to join a multicast group.
3. Verify that the internal IGMP table contains the multicast group information
4. Send a UDP data packet to the multicast group. Verify the socket is able to receive the data packet.
5. Transmit a response to the sender.
6. Use setsockopt to drop the multicast group. Verify that the internal IGMP table has dropped the group information.
7. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
   1. Basic BSD Raw Blocking Test. This scenario verifies the BSD raw socket option over the IPv4 and IPv6 network. All sockets are created as blocking sockets.

Test Procedure:

1. Initialize ip\_0 with one IPv4 address and one IPv6 global address.
2. Create a raw socket on IPv4 network for protocol type 100
3. Verify the raw socket is able to receive a raw packet with protocol 100. For IPv4 traffic, verify that that the IP header is present in the data.
4. Transmit a response to the sender.
5. Verify the sender is able to receive the data.
6. Close the raw socket.
7. Create a raw socket on IPv6 network for protocol 100
8. Repeat (3) – (6)
9. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
   1. Basic BSD RAW Non-Blocking Test. This scenario verifies the BSD raw socket option over the IPv4 and IPv6 network. All sockets are created as non-blocking sockets, and the test application uses select() to wait for events on the non-blocking sockets.

Test Procedure:

1. Initialize ip\_0 with one IPv4 address and one IPv6 global address.
2. Create a raw socket on IPv4 network for protocol type 100
3. Use fcntl to set the socket to non-blocking
4. Call recvfrom, verify that the recvfrom returns -1, and errno is EWOULDBLOCK
5. Select on the socket for read
6. After select returns, do another recvfrom
7. Verify that data is received. For IPv4 traffic, verify that that the IP header is present in the data.
8. Transmit a response to the sender
9. Close the socket
10. Create a raw socket on IPv6 network for protocol 100
11. Repeat (3)-(9)
12. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
    1. Basic BSD UDP Blocking Test. This scenario verifies the basic BSD UDP socket operations over IPv4 and IPv6 network, as client and as server. All sockets are created as blocking sockets.

Test Procedure:

1. Initialize ip\_0 with one IPv4 address and one IPv6 global address.
2. Create a UDP socket on the IPv4 network
3. Bind to a local port number and INADDR\_ANY
4. Verify that the UDP socket is able to receive data with recvfrom()
5. Verify that the server is able to transmit a response message to the sender using the remove address returned from the recvfrom() call.
6. Close the socket
7. Create a UDP socket on the IPv6 network
8. Repeat (3) – (6)
9. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
   1. Basic BSD UDP Non-Blocking Test. This scenario verifies the basic BSD UDP socket operations over IPv4 and IPv6 network, as client and as server. All sockets are created as non- blocking sockets. Test applications use select to wait for events on the non-blocking sockets.

Test Procedure:

1. Initialize ip\_0 with one IPv4 address and one IPv6 global address
2. Create a UDP socket on the IPv4 network
3. Use fcntl to set the socket to non-blocking
4. Call recvfrom, verify that the recvfrom returns -1, and errno is EWOULDBLOCK
5. Select on the socket for read
6. After select returns, do another recvfrom
7. Verify that data is received
8. Transmit a response to the sender
9. Close the socket
10. Create a UDP socket on IPv6 network
11. Repeat (3) – (9)
12. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
    1. Basic BSD TCP Non-Blocking Test. This scenario tests the behavior of BSD TCP sockets over both IPv4 and IPv6 network, as server sockets, and as client sockets. In this test scenario, all BSD sockets are created as non-blocking socket. The test application uses select to wait for events on each of the sockets. For each TCP connection, a new helper thread is created to handle the connection. Over each connection, the socket first receives a simulated request message followed by sending a response message. Both ends verify the connection with the expected peer, the content of each message passed.
13. Initialize ip\_0 and configure one IPv4 address and one IPv6 link local address, one IPv6 global address.
14. Create a TCP socket on the IPv4 network.
15. Use fcntl to configure the socket to be non-blocking
16. Connect to a TCP server
17. Verify that the connect would return -1 immediately, and errno is EINPROGRESS
18. Use select on WRITE\_FD set to wait for the connection to complete.
19. Call connect again and verify the return value is 0.
20. Send data
21. Close the socket.
22. Create a TCP socket on the IPv4 network
23. Use fcntl to configure the socket to be non-blocking
24. Bind to local port and INADDR\_ANY
25. Call listen
26. Select on the server socket. After select returns, call accept and verify the return value being 0
27. For each new socket, put the socket into nonblocking mode, and create a thread to handle the new connection.
    1. The helper thread select on the socket for READ.
    2. Select returns, and the helper thread calls “recv” to read the data
    3. The helper thread calls “send” to transmit a response message.
    4. The helper thread closes the new socket and exits from the thread.
28. After received multiple clients (10 clients in this test) the server socket is closed.
29. Create a TCP socket on the IPv6 network
30. Repeat (11) – (16)
31. Create TCP socket for the IPv4 network
32. Put the socket into non-blocking mode
33. Attempt to connect to an invalid port-address
34. Verify the connect call should return -1 immediately with errno set to EINPROGRESS
35. Use select to wait for the socket
36. Upon select returns on the socket, do another connect.
37. Verify that connect returns -1, and errno is ECONNREFUSED
38. Close the socket.
39. Repeat (19) – (26) except in (21) connect to an invalid IP address
40. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.

2.13 TCP 2nd Bind Test. This scenario verifies that two TCP sockets bind to the same port number and same interface address (INADDR\_ANY) would cause the 2nd bind to fail with errno set to EADDRINUSE.

Test Procedures:

1. Initialize ip\_0 with one IPv4 address
2. Create a TCP socket
3. Bind the TCP socket to local port 12345, interface INADDR\_ANY
4. Verify the return value being 0.
5. Create a 2nd TCP socket
6. Bind the 2nd TCP socket to the same port 12345, interface INADDR\_ANY
7. Verify that the return value is -1, and errno set to EADDRINUSE
8. Close both sockets
9. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.

2.14 TCP UDP select test: The scenario verifies that when both TCP socket and UDP socket has data for receiving, the select call is able to return indicating both sockets are readable.

Test Procedures:

1. Initialize ip\_0 and ip\_1 with IPv4 addresses.
2. Create a TCP socket, bind it to a local port and INADDR\_ANY
3. Issue a listen on the TCP socket
4. Create a TCP client thread, attempt to connect to the TCP server
5. Wait for accept on the TCP socket
6. Once the TCP socket is connected, create UDP socket
7. Bind the UDP socket to the local port and INADDR\_ANY
8. Sleep for one second
9. During this time, create a UDP client thread
10. From the TCP client thread a piece of data is sent to the TCP server
11. From the UDP client thread, a piece of data is sent to the UDP server
12. After the TCP/UDP server thread wakes up, it creates an FD\_SET for reading on the client TCP connection socket, and the UDP serversocket.
13. Performs a select
14. Verify that select returns 2
15. Verifies that both the UDP server socket is selected
16. Verify that data is available on the UDP server socket, and data matches the ones sent by the client
17. Verify that data is available on the TCP client socket, and data matches the ones sent by the client.
18. Disconnect the TCP client socket
19. Close the TCP client socket, the TCP server socket, and the UDP socket.
20. Verify that after this test scenario, all NetX socket control blocks, all BSD socket control blocks, and all NetX packets are properly released.
21. Test Report

Copyright (c) 1996-2012 Express Logic Inc. \* NetX Duo ARM9/IAR Version G5.6.5.1 SN: Internal\_Test\_05-08-12 \*

NetX Test: Basic BSD TCP UDP Select Test.................SUCCESS!

NetX Test: Basic BSD TCP 2nd Bind Test...................SUCCESS!

NetX Test: Basic BSD Raw Blocking Test...................SUCCESS!

NetX Test: Basic BSD RAW Non-Blocking Test...............SUCCESS!

NetX Test: Basic BSD TCP Basic Blocking Test.............SUCCESS!

NetX Test: Basic BSD TCP Multiple Accept Test............SUCCESS!

NetX Test: Basic BSD UDP Connect Test....................SUCCESS!

NetX Test: Basic BSD TCP Bind Test.......................SUCCESS!

NetX Test: Basic BSD UDP Bind Test.......................SUCCESS!

NetX Test: Basic BSD Multicast Test......................SUCCESS!

NetX Test: Basic BSD UDP Blocking Test...................SUCCESS!

NetX Test: Basic BSD UDP Non-Blocking Test...............SUCCESS!

NetX Test: Basic BSD TCP Disconnect Test.................SUCCESS!

NetX Test: Basic BSD TCP Non Blocking Connect Test.......SUCCESS!

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

\*\*\*\* Test Summary: Tests Passed: 14 Tests Failed: 0