Maxwell Pro Release: 2016a Build 16020 20-Jan-2016
Report Generated Wed Jan 20 16:53:11 2016

DUT IPv4 address: 10.10.10.11
DUT IPv6 address: fd28:6f7a:7ccb:0ee9::11
Maxtap IPv4 address/subnet: 10.10.10.70/32
Maxtap IPv6 address/subnet: fd28:6f7a:7ccb:0ee9::70/64

Type icmp is enabled: icmp.v4 tested icmp.v6 tested

- Seconds to wait for DUT response: 1
- Unfragmented Generated Echo Payload Length (icmp): 1300
- File to execute prior to each test:
- Alternate Traffic Source Program:

Type icmpb is enabled: icmpb.v4 tested icmpb.v6 tested

- Seconds to wait for DUT response: 2
- Fragmented Generated Echo Payload Length (icmpb): 9000
- File to execute prior to each test:
- Alternate Traffic Source Program:

Type tcp is enabled: tcp.v4 tested tcp.v6 tested

- DUT TCP server port: 5006
- Seconds to wait for DUT response: 1
- File to execute prior to each test:
- Alternate Traffic Source Program:
- Using Default Echo Exchange Payload
- Length: 9000

Type tcps is disabled

Type udp is enabled: udp.v4 tested udp.v6 untested

- DUT UDP Port: 5006
- Seconds to wait for DUT response: 1
- File to execute prior to each test:
- Alternate Traffic Source Program:
- Using Default Echo Exchange Payload
- Length: 1300

Type udpb is enabled: udpb.v4 tested udpb.v6 untested

- DUT UDP Port: 5006
- Seconds to wait for DUT response: 1
- File to execute prior to each test:

- Alternate Traffic Source Program:
- Using Default Echo Exchange Payload
- Length: 9000

Type udps is disabled

Type dhcpc is disabled

Type tls is disabled

Type tlss is disabled

Runs Requested	Runs Uncompleted	Passed Results	Failed Results	Ungraded Results	Skipped Tests
1	0	311	102	22	254

Calibration.Calibration.000 :

- **Calibration test to establish which traffic sources the DUT will echo.**
- This is the calibration and communication test used by the GUI automated test system. It collects ethernet addresses that are used by the graders for all other tests. It also establishes that the DUT responds to all the configured and enabled traffic source programs by running each program. If the GUI gets a response then the checkmark in all the tests that use that traffic program are checked. But if it doesn't get a response then it reports a failure and unchecks that program for all tests. You may override the calibration settings by manually checking/unchecking the program checkmarks for this test, which will cascade to all tests. Or you may override an individual test by checking/unchecking the desired traffic program.
- **Assumptions:**
  The DUT must be running the application that is expected to reply to the Maxwell generated traffic for the enabled traffic source programs. You should have disabled (via "Configure Traffic Sources") all traffic sources your DUT cannot support.
- **Expectations:**
  The DUT must respond to the traffic sources that you have configured and enabled.
- **References:**
  Maxwell Automated TCP/IP Test guide.

IPv4.Datagram.000 **Pass**

- **Test to insure the DUT is in a sane state.**
- This is a 'sanity' check. During multiple automated test runs the impairments may cause the DUT stack or echo servers to enter unresponsive states. This test passes traffic through unimpaired so that if

the DUT has entered an unresponsive state one or more failures responses will be returned, indicating one of the tests since the last successful sanity check caused the DUT to fail. The tests since that last successful sanity check will need to be manually analyzed to determine the test that caused the DUT to enter the 'bad' state.

- **Assumptions:**
  The DUT should be running the echo servers or other applications that are expected to reply to Maxwell generated traffic for the desired set of tests.
- **Expectations:**
  The DUT is expected to reply to the traffic sources with the responses defined for them in the XML sources files.
- **References:**
  RFC 1122: section 3.1 {Implement IP and ICMP}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.
  PASS: icmp.v6 got echo response, which is an expected result.
  PASS: icmpb.v6 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: udpb.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.
  PASS: tcp.v6 got echo response, which is an expected result.

IPv4.Datagram.001 **Pass**

- **Set protocol field to a value not yet assigned by IANA.**
- This test replaces the protocol field value of the IPv4 datagram header with a value (254) that has not been assigned by the IANA.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST drop the packet and SHOULD issue an ICMPv4 message of type 3, code 2: protocol unreachable.
- **References:**
  RFC 792: page 4
  RFC 1122: section 3.2.2.1 {Generate Dest Unreachable (code 2/3)}
- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.002 **Pass**

- **Set total length field so it goes past end of link-layer by 1.**
- This test sets the total length field of the IPv4 datagram such that the end of the datagram appears to extend past the end of the link-layer frame by

at least 1 byte. The datagram is not marked as a fragment.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 792: page 7
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.003 **Pass**

- **Set total length field so it goes past end of link-layer by 8.**
- This test sets the total length field of the IPv4 datagram such that the end of the datagram appears to extend past the end of the link-layer frame by at least 8 bytes. The datagram is not marked as a fragment.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 792: page 7
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.004 **Pass**

- **Set total length field to zero.**
- This test sets the total length field of the IPv4 datagram such that the datagram appears to contain no data but does not truncate the actual datagram content.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 792: page 7

RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}

- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.005 **Pass**

- **Set total length field to maximum size.**
- This test sets the total length field of the IPv4 datagram such that the datagram appears to contain 65535 (0xffff) bytes, but no additional bytes are actually added to the datagram.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 792: page 7
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.006 **Pass**

- **Set total length field to 19.**
- This test sets the payload length field of the IPv4 datagram such that the total length is less than the allowed minimum. But no bytes are actually removed from the original frame and the checksum is computed over the original range of bytes.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 791: pages 11, 13
  RFC 792: page 7
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.007 **Pass**

- **Set IHL field to 0.**
- This test sets the Internet Header Length (IHL) field to zero. The checksums are computed over the original range of bytes.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST notice the bad length and drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 791: page 11
  RFC 792: page 7
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 returned "No echo response", which is an expected result.
  PASS: udp.v4 returned "No echo response", which is an expected result.
  PASS: tcp.v4 returned "No echo response", which is an expected result.

IPv4.Datagram.008 **Pass**

- **Set IHL and total length fields so that header goes past end of datagram.**
- This test sets the Internet Header Length (IHL) to 15 (so that the header length is 60 bytes) and total length field to 40. As a result, the two field values make the end of the header appear to extend past the end of the datagram. The IP options area is filled with zeroed data bytes while all the other original frame data is untouched except that checksums are updated for the new total length.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST notice the bad length and drop the packet and SHOULD respond with an ICMPv4 message of type 12, code 0: parameter problem.
- **References:**
  RFC 791: page 11
  RFC 792: page 7
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.009 **Pass**

- **Change destination address to the loopback '127.0.0.1' address.**

- The destination address of the IPv4 packets is set to the loopback ('127.0.0.1') address.
- **Assumptions:**
  None.
- **Expectations:**
  Packets with a loopback 127.0.0.1 destination address MUST never appear outside a node and therefore interfaces MUST silently drop those incoming packets.
- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad dest addr}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.010 **Pass**

- **Change UDPv4 destination address to '0.0.0.0' broadcast address.**
- The destination address of the IPv4 packets is set to the non-standard broadcast ('0.0.0.0') address. The link-layer destination address is set to that of the DUT rather than to the Ethernet or 802 broadcast address. The RFCs do not disallow this combination.
- **Assumptions:**
  None.
- **Expectations:**
  RFC 1122, section 4.1.1 states: "UDP is used by applications ... that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP." Section 3.3.6 indicates hosts SHOULD recognize 0.0.0.0 as a destination broadcast address. Therefore by implication hosts SHOULD accept UDPv4 datagrams with a destination of 0.0.0.0 if they also accept 255.255.255.255 broadcast addresses.
- **References:**
  RFC 1122: section 3.3.6 {Receive 0 or -1 broadcast formats OK}
  RFC 1122: section 3.3.6 {Recognize all broadcast address formats}
  RFC 1122: section 4.1.1
- **Last Run Results:**
  PASS: udp.v4 got echo response, which is an expected result.

IPv4.Datagram.011 **Pass**

- **Change TCPv4 destination address to '0.0.0.0' broadcast address.**
- The destination address of the IPv4 packets is set to the non-standard broadcast ('0.0.0.0') address.
- **Assumptions:**
  None.
- **Expectations:**

TCP SYN segments with the destination address set to any broadcast address MUST be discarded.

- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad dest addr}
  RFC 1122: section 4.2.3.10 {Silently discard SYN to bcast/mcast addr}
- **Last Run Results:**
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.012 **Fail**

- **Change ICMPv4 destination address to '0.0.0.0' broadcast address.**
- The destination address of the IPv4 packets is set to the non-standard broadcast ('0.0.0.0') address.
- **Assumptions:**
  None.
- **Expectations:**
  ICMPv4 echo requests with the destination address set to any broadcast address MAY be silently discarded. However, there are valid rationale for accepting such requests - though the default for this test is to treat it as an error if they are not discarded.
- **References:**
  RFC 1122: section 3.2.2.6 {Discard Echo Request to broadcast address}
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
- **Last Run Results:**
  FAIL: icmp.v4 got echo response, which is not an expected result.

IPv4.Datagram.013 **Pass**

- **Change UDPv4 destination address to '255.255.255.255' broadcast address.**
- The destination address of the IPv4 packets is set to the non-standard broadcast ('255.255.255.255') address. The link-layer destination address is set to that of the DUT rather than to the Ethernet or 802 broadcast address. The RFCs do not disallow this combination.
- **Assumptions:**
  None.
- **Expectations:**
  RFC 1122, section 4.1.1 states: "UDP is used by applications ... that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP.strongly". Absent a prohibition in an RFC somewhere, we expect UDP to respond to destination broadcast address.
- **References:**
  RFC 1122: section 3.3.6 {Receive 0 or -1 broadcast formats OK}
  RFC 1122: section 3.3.6 {Recognize all broadcast address formats}
  RFC 1122: section 4.1.1
  RFC 1122: section 3.2.1.3

- **Last Run Results:**
  PASS: udp.v4 got echo response, which is an expected result.

IPv4.Datagram.014 **Pass**

- **Change TCPv4 destination address to '255.255.255.255' broadcast address.**
- The destination address of the IPv4 packets is set to the broadcast ('255.255.255.255') address.
- **Assumptions:**
  None.
- **Expectations:**
  TCP SYN segments with the destination address set to the standard broadcast address MUST be discarded.
- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad dest addr}
  RFC 1122: section 4.2.3.10 {Silently discard SYN to bcast/mcast addr}
- **Last Run Results:**
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.015 **Fail**

- **Change ICMPv4 destination address to '255.255.255.255' broadcast address.**
- The destination address of the IPv4 packets is set to the broadcast ('255.255.255.255') address.
- **Assumptions:**
  None.
- **Expectations:**
  ICMPv4 echo requests with the destination address set to any broadcast address MAY be silently discarded. However, there are valid rationale for accepting such requests - though the default for this test is to treat it as an error if they are not discarded.
- **References:**
  RFC 1122: section 3.2.2.6 {Discard Echo Request to broadcast address}
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
- **Last Run Results:**
  FAIL: icmp.v4 got echo response, which is not an expected result.

IPv4.Datagram.016 **Pass**

- **Change source address to the '0.0.0.0' broadcast address.**
- The source address of the IPv4 packets is set to the non-standard ('0.0.0.0') broadcast address.
- **Assumptions:**
  None.

- **Expectations:**
  Packets with a broadcast source address MUST be silently discarded by hosts. This is only a valid source address in an DHCP/BOOTP address request.
- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad src addr}
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
  RFC 1122: section 4.2.3.10 {Reject SYN from invalid IP address}
  RFC 1122: section 4.1.3.6 {Bad IP src addr silently discarded by UDP/IP}
  RFC 1122: section 4.1.3.6 {Only send valid IP source address}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.017 **Pass**

- **Change source address to the loopback '127.0.0.1' address.**
- The source address of the IPv4 packets is set to the loopback ('127.0.0.1') address.
- **Assumptions:**
  None.
- **Expectations:**
  Packets with the destination address set to the loopback address MUST never appear outside a node and all nodes MUST silently drop any packets arriving on a node's interface.
- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad src addr}
  RFC 1122: section 4.2.3.10 {Reject SYN from invalid IP address}
  RFC 1122: section 4.1.3.6 {Bad IP src addr silently discarded by UDP/IP}
  RFC 1122: section 4.1.3.6 {Only send valid IP source address}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.018 **Pass**

- **Change source address to the '255.255.255.255' broadcast address.**
- The source address of the IPv4 packets is set to the broadcast ('255.255.255.255') address.
- **Assumptions:**
  None.
- **Expectations:**

Packets with broadcast source addresses are not allowed and all such packets MUST be silently dropped.

- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad src addr}
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
  RFC 1122: section 4.2.3.10 {Reject SYN from invalid IP address}
  RFC 1122: section 4.1.3.6 {Bad IP src addr silently discarded by UDP/IP}
  RFC 1122: section 4.1.3.6 {Only send valid IP source address}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.019 **Pass**

- **Set *protocol* field to 200 (unassigned) and source address to the non-standard '0.0.0.0' broadcast address.**
- The *protocol* field of the IPv4 packets is set to the value 200 (which is not yet assigned by the IANA) and the source address of the packets is set to the non-standard '0.0.0.0' broadcast address.
- **Assumptions:**
  None.
- **Expectations:**
  A node would normally issue an ICMPv4 protocol unreachable message back to the source when an unknown protocol value is encountered. But if the source address is not a specific address the datagram must be silently dropped.
- **References:**
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
  RFC 1122: section 3.2.2 {IP b'cast or IP m'cast}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
  RFC 1122: section 3.2.2 {Datagram with non-unique src address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.020 **Pass**

- **Set *protocol* field to 200 (unassigned) and source address to the '127.0.0.1' loopback address.**
- The *protocol* field of the IPv4 packets is set to the value 200 (which is not yet assigned by the IANA) and the source address of the packets is set to the loopback ('127.0.0.1') address.
- **Assumptions:**

None.
- **Expectations:**
  A node would normally issue an ICMPv4 protocol unreachable message back to the source when an unknown protocol value is encountered. But if the source address is the loopback address the datagram must be silently dropped.
- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad src addr}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
  RFC 1122: section 3.2.2 {Datagram with non-unique src address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.021 **Pass**

- **Set *protocol* field to 200 (unassigned) and source address to the '255.255.255.255' broadcast address.**
- The *protocol* field of the IPv4 packets is set to the value 200 (which is not yet assigned by the IANA) and the source address of the packets is set to the broadcast ('255.255.255.255') address.
- **Assumptions:**
  None.
- **Expectations:**
  A node would normally issue an ICMPv4 protocol unreachable message back to the source when an unknown protocol value is encountered. But if the source address is not a specific address the datagram must be silently dropped.
- **References:**
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
  RFC 1122: section 3.2.2 {IP b'cast or IP m'cast}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
  RFC 1122: section 3.2.2 {Datagram with non-unique src address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.022 **Pass**

- **Set TTL field to zero and source address to the non-standard '0.0.0.0' broadcast address.**
- The TTL field of the IPv4 packets are set to zero and the source address of the packets are set to the non-standard ('0.0.0.0') broadcast address.
- **Assumptions:**

None.
- **Expectations:**
  The DUT MUST discard the datagram and MUST NOT issue any ICMP error messages (such as a Time Exceeded problem message.)
- **References:**
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
  RFC 1122: section 3.2.2 {IP b'cast or IP m'cast}
  RFC 1122: section 3.2.1.7 {Send packet with TTL of 0}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
  RFC 1122: section 3.2.2 {Datagram with non-unique src address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.023 **Pass**

- **Set TTL field to zero and source address to the '127.0.0.1' loopback address.**
- The TTL field of the IPv4 packets are set to zero and the source address of the packets are set to the loopback ('127.0.0.1') address.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST discard the datagram and MUST NOT issue any ICMP error messages (such as a Time Exceeded problem message.)
- **References:**
  RFC 1122: section 3.2.1.3 {Silently discard datagram with bad src addr}
  RFC 1122: section 3.2.1.7 {Send packet with TTL of 0}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
  RFC 1122: section 3.2.2 {Datagram with non-unique src address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.024 **Pass**

- **Set TTL field to zero and source address to the '255.255.255.255' broadcast address.**
- The TTL field of the IPv4 packets are set to zero and the source address of the packets are set to the broadcast ('255.255.255.255') address.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT MUST discard the datagram and MUST NOT issue any ICMP

error messages (such as a Time Exceeded problem message.)
- **References:**
  RFC 1122: section 3.2.1.3 {Broadcast addr as IP source addr}
  RFC 1122: section 3.2.2 {IP b'cast or IP m'cast}
  RFC 1122: section 3.2.1.7 {Send packet with TTL of 0}
  RFC 1122: section 3.2.1.3 {Src address must be host's own IP address}
  RFC 1122: section 3.2.2 {Datagram with non-unique src address}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.025 **Pass**

- **Truncate the datagram so protocol field indicates another protocol follows, but none does.**
- The total length field is set so it equals only the length covered by the header length. The checksum is updated so it is correct for the covered length. But the data making up the protocol that was indicated by the protocol field is not transmitted. The result is that the datagram should appear valid in all respects but is otherwise truncated.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must not attempt to parse upper layer protocol bytes that weren't sent. The packet must be discarded and an ICMP protocol problem message should be sent.
- **References:**
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.026 **Pass**

- **Truncate the datagram so protocol field indicates another protocol follows, but only 1 byte does.**
- The total length field is set so it equals only the length covered by the header length plus one. The checksum is updated so it is correct for the covered length. But only one byte of the data making up the protocol that was indicated by the protocol field is transmitted. The result is that the datagram should appear valid in all respects but is otherwise truncated. Please note that since the error condition will be noticed in the TCP processing layer, no ICMP error message should be issued. The requirement to send ICMP error messages in section 3.3.8 of RFC 1122

applies only to problems found in the IP layer.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must not attempt to parse upper layer protocol bytes that weren't sent. The packet must be discarded. An ICMP problem message is not applicable so should not be sent.
- **References:**
  RFC 1122: section 3.1
  RFC 1122: section 3.3.8 {Return ICMP error msgs (when not prohibited)}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.027 **Pass**

- **Change IPv4 version field to value other than 4.**
- The version field of the IPv4 headers is changed to the value of 5 in order to verify that the DUT silently discards them.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must silently discard the packets, so any ICMP error messages will be considered an error, as will any other response to the packets.
- **References:**
  RFC 1122: section 3.2.1.1 {Silently discard Version != 4}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: icmpb.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: udpb.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.028 **Pass**

- **Adjust header so checksum is -0 and set checksum field to -0.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  Two bytes of an unknown option are adjusted so that the checksum yields -0. Since it is the complement of the checksum (not the checksum itself!) that is stored in the checksum field, the checksum field can therefore never contain all ones (see discussion section of RFC 1624.) But instead of +0 in the checksum field, -0 is stored. However, the ones complement sum check must still yield all ones, indicating no transmission errors.

- **Assumptions:**
  None.
- **Expectations:**
  The DUT must send an echo reply.
- **References:**
  RFC 1122: section 3.2.1.2 {Verify IP checksum, silently discard bad dgram}
  RFC 791: section 3.1
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.Datagram.029 **Pass**

- **Adjust header so checksum is -0 and set checksum field to +0.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  Two bytes of an unknown option are adjusted so that the checksum yields -0. Since it is the complement of the checksum (not the checksum itself!) that is stored in the checksum field, the checksum field can therefore never contain -0 (see discussion section of RFC 1624.)
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must send an echo reply.
- **References:**
  RFC 1122: section 3.2.1.2 {Verify IP checksum, silently discard bad dgram}
  RFC 791: section 3.1
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.

IPv4.Datagram.030 **Pass**

- **Adjust header so checksum is +1 and set checksum field to +0.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  Two bytes of an unknown option the are adjusted so that the checksum yields +1. However, the checksum field is set to +0.
- **Assumptions:**
  None.
- **Expectations:**

The DUT must silently reject the echo requests.
- **References:**
  RFC 1122: section 3.2.1.2 {Verify IP checksum, silently discard bad dgram}
  RFC 791: section 3.1
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: icmpb.v4 got no echo response, which is an expected result.

IPv4.Datagram.031 **Pass**

- **Set checksum field so it contains -1.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  The checksum field of the first data segment is set to -1 (0xFFFE in hex) but the header is not adjusted to yield a checksum of +1.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must silently reject the echo requests.
- **References:**
  RFC 1122: section 3.2.1.2 {Verify IP checksum, silently discard bad dgram}
  RFC 791: section 3.1
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: icmpb.v4 got no echo response, which is an expected result.

IPv4.Datagram.032 **Pass**

- **Invert the checksum field bit values.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  The checksum field bits are all inverted.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must silently reject the echo requests.
- **References:**
  RFC 1122: section 3.2.1.2 {Verify IP checksum, silently discard bad dgram}
  RFC 791: section 3.1

RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: icmpb.v4 got no echo response, which is an expected result.

IPv4.Datagram.033 **Pass**

- **Verify processing of TTL field.**
- This test sets the TTL field to zero for packets sent to the DUT. It then examines the TTL field for packets coming from the DUT to insure that their value is greater than zero.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must accept the datagrams with TTL of zero and must set the TTL value of its datagrams to a value greater than zero.
- **References:**
  RFC 1122: section 3.2.1.7 {Send packet with TTL of 0}
  RFC 1122: section 3.2.1.7 {Discard received packets with TTL < 2}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: udpb.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Datagram.034 **Pass**

- **Change ICMPv4 destination address to '224.0.0.1' multicast address.**
- The destination address of the IPv4 packets is set to the 'All Hosts' multicast address ('224.0.0.1').
- **Assumptions:**
  Echo responses to the 'All Hosts' multicast address needs to be enabled if the DUT allows for it.
- **Expectations:**
  ICMPv4 echo requests with the destination address set to any multicast address may be silently discarded. This test grades a response as a pass since a multicast echo response is needed to verify that the DUT joined the all-hosts group at startup, per RFC 1122.
- **References:**
  RFC 1122: section 3.2.2.6 {Discard Echo Request to multicast address}
  RFC 1122: section 3.3.7 {Join all-hosts group at startup}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.Datagram.035 **Pass**

- **Check for continuous dead gateway detection pinging.**
- This test listens for pings on the default gateway to verify that the DUT does not try to detect dead gateway by peridic pinging (ICMPv4 echo requests.) It listens for 120 seconds and yields a pass if no pings are detected in that time frame. Obviously this procedure will not detect active pinging that uses periods longer than 120 seconds, but it should provide some medium level of confidence for black box testing of IPv4 stacks.
- **Assumptions:**
  The IPv4 address of the default gateway must be specified so that pings on it can be detected.
- **Expectations:**
  Any ICMPv4 echo request on the default gateway address yields a failing grade.
- **References:**
  RFC 1122: section 3.3.1.4 {Ping gateways continuously}
- **Last Run Results:**
  PASS: The DUT sent 0 echo requests to the gateway address. The limit was 0 echo requests.

IPv4.Datagram.036 **Pass**

- **Check for dead gateway detection pinging while traffic being sent.**
- This test establishes a TCP connection with the DUT via the default gateway. It starts dropping segments from the traffic source after the inital SYN to simulate a possible gateway failure. The test listens for pings from the DUT to the default gateway to verify that the DUT does not send an excessive number of pings (ICMP echo requests) to detect dead gateway. It listens for 120 seconds and yields a pass if no more than 3 pings are detected in that time frame.
- **Assumptions:**
  The IPv4 address of the default gateway must be specified so that pings on it can be detected.
- **Expectations:**
  More than three ICMPv4 echo request on the default gateway address yields a failing grade.
- **References:**
  RFC 1122: section 3.3.1.4 {Ping only when traffic being sent}
  RFC 1122: section 3.3.1.4 {Ping only when no positive indication}
- **Last Run Results:**
  PASS: The DUT sent 0 echo requests to the gateway address. The limit was 3 echo requests.

IPv4.Datagram.037 **Fail**

- **Verify link layer broadcasts with specific addresses are discarded.**
- This test changes the Ethernet destination address of IPv4 datagrams sent to the DUT to the broadcast (FF:FF:FF:FF:FF:FF) address. Since the destination IPv4 address is not a multicast or broadcast address, the DUT should discard the datagrams.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should silently discard the datagrams.
- **References:**
  RFC 1122: section 3.3.6 {Silently discard link-layer-only b'cast dg's}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: icmpb.v4 got no echo response, which is an expected result.
  FAIL: udp.v4 got echo response, which is not an expected result.
  FAIL: udpb.v4 got echo response, which is not an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Datagram.038 **Fail**

- **Source address of reply from multihomed host matches destination address of request.**
- When a DUT has multiple interfaces (multihomed) and an IPv4 datagram request arrives that requires a response, the DUT should set the source address field to the address that was in the destination address of the request datagram. This test requires the DUT to have at least two interfaces, with the second interface having an IPv4 address one larger than the first. For example, if the DUT's first interface has an address of 10.0.0.4, then the second interface must be assigned 10.0.0.5.
  The test sets the IPv4 destination address of the request datagrams to the address of the second interface and expects a response from that interface. The Ethernet destination address of the request datagrams are set to the first interface.This combination should cause request datagrams to appear at the first DUT interface and reply datagrams to appear from the second interface.
- **Assumptions:**
  The DUT must be multihomed and two of the interfaces must be connected to a bridge or hub (but not a router.) The Maxwell interface must also be connected to the bridge. The IPv4 address assigned to the second DUT interface must be one greater than the first DUT interface. For example, if the first DUT interface has been assigned address 10.0.0.3, then the second interface must be assigned 10.0.0.4. The DUT should be set to use the first interface as the default route.
- **Expectations:**
  The DUT should send reply datagrams from the second DUT interface.
- **References:**

RFC 1122: section 3.3.4.2 {Reply with same addr as spec-dest addr}

- **Last Run Results:**
  FAIL: icmp.v4 got no echo response, which is not an expected result.
  FAIL: icmpb.v4 got no echo response, which is not an expected result.
  FAIL: udp.v4 got no echo response, which is not an expected result.
  FAIL: udpb.v4 got no echo response, which is not an expected result.
  FAIL: tcp.v4 got no echo response, which is not an expected result.

IPv4.Datagram.039 **Fail**

- **Verify that ICMPv4 host and net redirects are treated the same.**
- This test checks that ICMPv4 redirects operate as expected. It does this by forcing the DUT to reply to an ICMP echo request via its default gateway. On getting the DUT's first echo reply, an ICMP NETWORK REDIRECT is sent that changes the gateway to an alternate IP address and MAC address. A second echo request is sent and the echo reply destination is expected to be to the alternate address. An ICMP HOST REDIRECT is then sent that changes the gateway to yet another alternate IP and MAC address. Another echo request is sent and the echo reply destination is expected to be the latest alternate gateway. Final ICMP NETWORK and HOST REDIRECTs are sent that return the routes to the original gateway. The test sets the source address of the echo request datagrams to 74.125.224.242. It increments the last byte of the original gateway addresses in order to form the temporary IP and MAC redirect addresses.
- **Assumptions:**
  An address must be entered into the 'Router IPv4' field, otherwise the test is skipped. The host byte of the IPv4 address is incremented by 1, 2, and 3 to create alternate router addresses, so those addresses must also be unused by other systems so Maxwell can use them. Lastly, The DUT must also be configured to accept redirects. Some systems have redirects disabled by default as a security precaution, or they also have lists of acceptable routers. These need to be disabled.
- **Expectations:**
  For each echo request, the DUT must send its echo replies to the last redirected gateway.
- **References:**
  RFC 1122: section 3.2.2.2 {Update route cache when recv Redirect}
  RFC 1122: section 3.2.2.2 {Handle both Host and Net Redirects}
  RFC 1122: section 3.3.1.2 {Treat Host and Net Redirect the same}
  RFC 1122: section 3.3.1.5 {Switch from failed default g'way to another}
  RFC 792: page 12
- **Last Run Results:**
  FAIL: Network redirect failed. Host redirect failed.

IPv4.Fragment.000 **Pass**

- **Test to insure the DUT is in a sane state.**
- This is a 'sanity' check. During multiple automated test runs the impairments may cause the DUT stack or echo servers to enter unresponsive states. This test passes traffic through unimpaired so that if the DUT has entered an unresponsive state one or more failures responses will be returned, indicating one of the tests since the last successful sanity check caused the DUT to fail. The tests since that last successful sanity check will need to be manually analyzed to determine the test that caused the DUT to enter the 'bad' state.
- **Assumptions:**
  The DUT should be running the echo servers or other applications that are expected to reply to Maxwell generated traffic for the desired set of tests.
- **Expectations:**
  The DUT is expected to reply to the traffic sources with the responses defined for them in the XML sources files.
- **References:**
  None applicable.
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.
  PASS: icmp.v6 got echo response, which is an expected result.
  PASS: icmpb.v6 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: udpb.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.
  PASS: tcp.v6 got echo response, which is an expected result.

IPv4.Fragment.001 **Pass**

- **Fragment: 512 MTU. Order: 1 2 3 ... N. Overlap: None.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

PASS: udp.v4 got echo response, which is an expected result.
PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.002 **Pass**

- **Fragment: 512 MTU. Order: N ... 3 2 1. Overlap: None.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of decreasing segment offset, i.e. the last fragment goes first then the penultimate fragment etc.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.003 **Pass**

- **Fragment: 512 MTU. Order: 2 1 3 4 ... N. Overlap: None.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The MTU is 512. A final short fragment may occur at the end. The order of the first and second fragment (if any) are reversed on transmission, as might occur if the first fragment were delayed due to an ARP. The remaining fragments, if any, are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}

- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.004 **Fail**

- **Fragment: 8 MTU. Order: 1 2 3 ... N. Overlap: None.**
- Datagrams are fragmented into minimal sized chunks (IP header plus 8 bytes of data), starting at the start of the datagram. A final short fragment may occur at the end. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  FAIL: icmp.v4 got no echo response, which is not an expected result.
  FAIL: udp.v4 got no echo response, which is not an expected result.
  FAIL: tcp.v4 got no echo response, which is not an expected result.

IPv4.Fragment.005 **Pass**

- **Fragment: 512 MTU. Order: 1 2 3 ... N. Overlap: By 8 bytes.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The fragments overlap with fragment N+1 overlapping the end of fragment N by 8 bytes (one Network Fragmentation Block). E.g. Here is a sequence of NFBs in the fragments each holding 6 NFBs:

```
pkt 0: ABCDEF
pkt 1:      FGHIJK
pkt 2:           KLMNOP
...
```

  The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.

- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.006 **Pass**

- **Fragment: 512 MTU. Order: 1 2 3 ... N. Overlap: By 16 bytes.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The fragments overlap with fragment N+1 overlapping the end of fragment N by 16 bytes (two Network Fragmentation Blocks). E.g. Here is a sequence of NFBs in the fragments each holding 6 NFBs:

```
pkt 0: ABCDEF
pkt 1:     EFGHIJ
pkt 2:         IJKLMN
...
```

  The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.007 **Pass**

- **Fragment: 512 MTU. Order: 1 2 3 ... N. Overlap: All but 8 bytes.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The fragments overlap with fragment N+1 overlapping the

entire portion of fragment N-1 except for its first 8 bytes (one Network Fragmentation Block). E.g. Here is a sequence of NFBs in the fragments each holding 6 NFBs:

```
pkt 0: ABCDEF
pkt 1:  BCDEFG
pkt 2:   CDEFGH
...
```

The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of increasing segment offset.

- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.008 **Pass**

- **Fragment: 6 fragments. Order: Complex. Overlap: Complex.**
- This is a mildly complex fragmention scenario. Six fragments are generated as follows (the sizes shown are approximations, the important part is the degree of overlap and non-overlap of the fragments):

```
Packet 1:  ABCDEF
Packet 2:          IJKLMN
Packet 3:                  QRSTUV
Packet 4:         HIJKLMNO
Packet 5:        GHIJKLMNO
Packet 6:         HIJKLMNOP
```

Packet 3 may be short. Fragments are transmitted in order 1, 2, 3, 4, 5, 6.

- **Assumptions:**
  This pattern is possible only if the unfragmented segment has at least 7 NFB's, i.e. has at least 56 bytes of IP data
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.

- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.009 **Pass**

- **Fragment: 4 fragments. Order: Complex. Overlap: Complex.**
- Four fragments are generated as follows (the sizes shown are approximations, the important part is the degree of overlap and non-overlap of the fragments):

```
Packet 1:  ABCDEF
Packet 2:         HIJKLM
Packet 3:                OPQRST
Packet 4:        GHIJKLMN
```

  Packet 3 may be short. Fragments are transmitted in order 1, 2, 3, 4.
- **Assumptions:**
  This pattern is possible only if the unfragmented packet has at least 5 NFB's, i.e. has at least 40 bytes of IP data
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.010 **Pass**

- **Fragment: 512 MTU. Order: N ... 3 2 1. Overlap: By 8 bytes.**
- Packets are fragmented into MTU sized chunks, starting at the front of the packet. The fragments overlap with fragment N+1 overlapping the end of fragment N by 8 bytes (one Network Fragmentation Block). E.g. Here is a sequence of NFBs in the fragments each holding 6 NFBs:

```
pkt 0: ABCDEF
pkt 1:      FGHIJK
pkt 2:           KLMNOP
...
```

The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of decreasing segment offset.

- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.011 **Pass**

- **Fragment: 512 MTU. Order: N ... 3 2 1. Overlap: By 16 bytes.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The fragments overlap with fragment N+1 overlapping the end of fragment N by 16 bytes (two Network Fragmentation Blocks). E.g. Here is a sequence of NFBs in the fragments each holding 6 NFBs:

```
pkt 0: ABCDEF
pkt 1:     EFGHIJ
pkt 2:         IJKLMN
...
```

The MTU is 512. A final short fragment may occur at the end. Fragments are transmitted in order of decreasing segment offset.

- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}

RFC 1122: section 3.3.2 {At least 576 byte datagrams}

- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.012 **Fail**

- **Fragment: 512 MTU. Order: N ... 3 2 1. Overlap: All but 8 bytes.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The fragments overlap with fragment N+1 overlapping the entire portion of fragment N-1 except for its first 8 bytes (one Network Fragmentation Block). E.g. Here is a sequence of NFBs in the fragments each holding 6 NFBs:

```
pkt 0: ABCDEF
pkt 1:  BCDEFG
pkt 2:   CDEFGH
...
```

  The MTU is set at 512. A final short fragment may occur at the end. Fragments are transmitted in order of decreasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  FAIL: icmp.v4 got no echo response, which is not an expected result.
  FAIL: udp.v4 got no echo response, which is not an expected result.
  FAIL: tcp.v4 got no echo response, which is not an expected result.

IPv4.Fragment.013 **Pass**

- **Fragment: 6 fragments. Order: Complex reversed. Overlap: Complex.**
- This is a mildly complex fragmention scenario. It is the same as test 050.010.9 but with the packets sent in the reverse order. Six fragments are generated as follows (the sizes shown are approximations, the important part is the degree of overlap and non-overlap of the fragments):

```
Packet 1:  ABCDEF
Packet 2:          IJKLMN
Packet 3:                  QRSTUV
Packet 4:         HIJKLMNO
Packet 5:        GHIJKLMNO
Packet 6:         HIJKLMNOP
```

Packet 3 may be short. Fragments are transmitted in order 6, 5, 4, 3, 2, 1.

- **Assumptions:**
  This pattern is possible only if the unfragmented packet has at least 7 NFB's, i.e. has at least 56 bytes of IP data
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.014 **Pass**

- **Fragment: 4 fragments. Order: Complex reversed. Overlap: Complex.**
- This is the same test as 050.010.10 except that the order of the packets is reversed. Four fragments are generated as follows (the sizes shown are approximations, the important part is the degree of overlap and non-overlap of the fragments):

```
Packet 1:  ABCDEF
Packet 2:         HIJKLM
Packet 3:                OPQRST
Packet 4:        GHIJKLMN
Packet 3 may be short. Fragments are transmitted in order 4, 3, 2, 1
```

- **Assumptions:**
  This pattern is possible only if the unfragmented packet has at least 5 NFB's, i.e. has at least 40 bytes of IP data
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}

RFC 1122: section 3.2.1.4 {Support reassembly}

- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.015 **Pass**

- **Fragment: 256 MTU. Order: 1 2 3 ... N 1. Overlap: None.**
- This test exercises the ability of the receiver to garbage collect buffers that exist as the result of stray duplicate fragments that arrive after the full IP datagram has been reassembled from previously arriving fragments. Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. We only approximate the desired MTU - we assume that the option size won't change when the option copy routine is called. The MTU is set at 256. A final short fragment may occur at the end. In this test we begin by sending all of the fragments and then re-sending the first fragment. Except for the retransmitted fragment, all fragments are transmitted in order of increasing fragment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer. The final first fragment must be discarded after the reassembly timeout (recommended between 60 and 120 seconds by RFC 1122) and an ICMP Time Exceeded message must be sent to the source host.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.016 **Pass**

- **Fragment: 256 MTU. Order: 1 2 3 ... N N. Overlap: None.**
- This test exercises the ability of the receiver to garbage collect buffers that exist as the result of stray duplicate fragments that arrive after the full IP datagram has been reassembled from previously arriving fragments. Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. We only approximate the desired MTU - we assume that the option size won't change when the option copy routine is called.

The MTU is set at 256. A final short fragment may occur at the end.
In this test we begin by sending all of the fragments and then re-sending the final fragment. Except for the retransmitted fragment, all fragments are transmitted in order of increasing fragment offset.

- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer. The repeated final fragment must be discarded after the reassembly timeout (recommended between 60 and 120 seconds by RFC 1122). An ICMP Time Exceeded message should NOT be sent.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.017 **Pass**

- **Fragment: 256 MTU. Order: 1 ... R R+2 ... N 1 ... N. Overlap: None.**
- This test exercises the ability of the receiver to garbage collect buffers that exist as the result of stray duplicate fragments that arrive after the full IP datagram has been reassembled from previously arriving fragments.
  Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. We only approximate the desired MTU - we assume that the option size won't change when the option copy routine is called.
  The MTU is set at 256. A final short fragment may occur at the end.
  In this test we send the original fragmented datagram twice. The first time we go through and send all fragments except for one randomly selected one. The second time we go through and send all the fragments. In both passes the fragments are transmitted in order of increasing fragment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}

RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.018 **Pass**

- **Fragment: 512 MTU. Order: 1 ... N. Overlap: None. Appends random NOPs.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The MTU is 512. A final short fragment may occur at the end. Each fragment is extended with a random number of No Operation IP options and an End-of-Option-List option. This means that the IP header could be of differing size on the different fragments. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.019 **Pass**

- **Fragment: 512 MTU. Order: 1 ... N. Overlap: None. Prepends random NOPs.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The MTU is 512. A final short fragment may occur at the end. The options for each fragment are prepended with a random number of No Operation IP options and (if needed) an End-of-Option-List option. This means that the IP header could be of differing size on the different fragments. It also means the options from the original datagram will overlap each other in different fragments, thus exercising whether the receiver simply overlays the options or intelligently assembles them. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**

None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer. Options from the fragments must be intelligently aggregated.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.020 **Pass**

- **Fragment: 512 MTU. Order: 1 ... N. Overlap: None. Appends NOPs.**
- Datagrams are fragmented into MTU sized chunks, starting at the front of the datagram. The MTU is 512. A final short fragment may occur at the end. Each fragment is extended with No Operation IP options that run to the end of the IP header without an End-of-Option-List option. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
  RFC 1122: section 3.3.2 {At least 576 byte datagrams}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Fragment.021 **Fail**

- **Fragment: 8 MTU. Order: N ... 3 2 1. Overlap: None.**
- Datagrams are fragmented into minimal sized chunks (IP header plus 8 bytes of data), starting at the start of the datagram.
  A final short fragment may occur at the end.

Fragments are transmitted in order of decreasing segment offset, i.e. the last fragment goes first then the penultimate fragment etc.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  FAIL: icmp.v4 got no echo response, which is not an expected result.
  FAIL: udp.v4 got no echo response, which is not an expected result.
  FAIL: tcp.v4 got no echo response, which is not an expected result.

IPv4.Fragment.022 **Pass**

- **Fragment: 'Ping of death.' 256 MTU. Order 1 ... N. Overlap: None.**
- This performs the impairment colloquially known as the 'ping of death'. It sends a series of fragments such that the end of the last fragment exceeds the longest possible datagram length (65535). The problem really affects IP, not ICMP, so the colloquial name is misleading. To perform the impairment the first 256 bytes of the incoming datagram is repeated into each manufactered fragment. Enough fragments are sent to create 65536 bytes of data in any destination buffer (one more that is allowed.) The MTU is 512 + header length. A final fragment of 8 bytes occurs at the end. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must eventually be discarded before anything is delivered to upper protocol layers. A parameter problem can be issued.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Fragment.023 **Pass**

- **Fragment: 'Ping of death.' 256 MTU. Order N ... 1. Overlap: None.**
- This performs the impairment colloquially known as the 'ping of death'. It sends a series of fragments such that the end of the last fragment exceeds the longest possible datagram length (65535). The problem really affects IP, not ICMP, so the colloquial name is misleading. To perform the impairment the first 256 bytes of the incoming datagram is repeated into each manufactered fragment. Enough fragments are sent to create 65536 bytes of data in any destination buffer (one more that is allowed.) The MTU is 512 + header length. A final fragment of 8 bytes occurs at the end. Fragments are transmitted in order of decreasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must eventually be discarded before anything is delivered to upper protocol layers. A parameter problem can be issued.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Fragment.024 **Pass**

- **Fragment: 'Ping of death.' 96 MTU. Order 1 ... N. Overlap: None.**
- This performs the impairment colloquially known as the 'ping of death'. It sends a series of fragments such that the end of the last fragment exceeds the longest possible datagram length (65535). The problem really affects IP, not ICMP, so the colloquial name is misleading. To perform the impairment the first 96 bytes of the incoming datagram is repeated into each manufactered fragment. Enough fragments are sent to create 65536 bytes of data in any destination buffer (one more that is allowed.) The MTU is 96 + header length. A final fragment of 8 bytes occurs at the end. Fragments are transmitted in order of increasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must eventually be discarded before anything is delivered to upper protocol layers. A parameter problem can be issued.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}

RFC 1122: section 3.2.1.4 {Support reassembly}

- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Fragment.025 **Pass**

- **Fragment: 'Ping of death.' 96 MTU. Order N ... 1. Overlap: None.**
- This performs the impairment colloquially known as the 'ping of death'. It sends a series of fragments such that the end of the last fragment exceeds the longest possible datagram length (65535). The problem really affects IP, not ICMP, so the colloquial name is misleading. To perform the impairment the first 96 bytes of the incoming datagram is repeated into each manufactered fragment. Enough fragments are sent to create 65536 bytes of data in any destination buffer (one more that is allowed.) The MTU is 96 + header length. A final fragment of 8 bytes occurs at the end. Fragments are transmitted in order of decreasing segment offset.
- **Assumptions:**
  None.
- **Expectations:**
  The fragments must eventually be discarded before anything is delivered to upper protocol layers. A parameter problem can be issued.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got no echo response, which is an expected result.

IPv4.Fragment.026 **Fail**

- **Fragment: 6 fragments, 6th missing. Order: Complex. Overlap: Complex.**

- This is a mildly complex fragmention scenario. Six fragments are generated as follows (the sizes shown are approximations, the important part is the degree of overlap and non-overlap of the fragments):

```
Packet 1:  ABCDEF
Packet 2:          IJKLMN
Packet 3:                 QRSTUV
Packet 4:         HIJKLMNO
Packet 5:        GHIJKLMNO
Packet 6:         HIJKLMNOP
```

  Packet 3 may be short. Fragments are transmitted in order 1, 2, 3, 4, and 5. However, packet 6 is not sent.
- **Assumptions:**
  This pattern is possible only if the unfragmented segment has at least 7 NFB's, i.e. has at least 56 bytes of IP data
- **Expectations:**
  Reassembly is not possible. The DUT is given 120 seconds to send a type 11, code 1 (fragment reassembly time exceeded.) The test fails if this does not happen in the time given.
- **References:**
  RFC 1122: section 3.3.2 {Send ICMP Time Exceeded on reassembly timeout}
- **Last Run Results:**
  FAIL: icmp.v4 got no echo response, which is not an expected result.

IPv4.Fragment.027 **Pass**

- **Fragment: 6 fragments, initial missing. Order: Complex. Overlap: Complex.**
- This is a mildly complex fragmention scenario. Six fragments are generated as follows (the sizes shown are approximations, the important part is the degree of overlap and non-overlap of the fragments):

```
Packet 1:  ABCDEF
Packet 2:          IJKLMN
Packet 3:                 QRSTUV
Packet 4:         HIJKLMNO
Packet 5:        GHIJKLMNO
Packet 6:         HIJKLMNOP
```

  Packet 3 may be short. Fragments are transmitted in order 1, 2, 3, 4, and 5. However, packet 1, the initial packet, is not sent.
- **Assumptions:**
  This pattern is possible only if the unfragmented segment has at least 7 NFB's, i.e. has at least 56 bytes of IP data
- **Expectations:**
  Reassembly is not possible. The DUT is given 120 seconds to send a type 11, code 1 (fragment reassembly time exceeded.) The test fails if the DUT

DOES send this error message.
- **References:**
  RFC 1122: section 3.3.2 {Send ICMP Time Exceeded on reassembly timeout}
  RFC 1122: section 3.2.2 {Non-initial fragment}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.

IPv4.Fragment.028 **Pass**

- **Implementation of local fragmentation.**
- Although the icmpb.v4 traffic source is used in several other tests, none of the exclusively depend on that traffic source being available. In order to positively ascertain that the local fragmentation of outgoing packets is implemented per RFC 1122, this test was created. If this test is skipped or fails then the DUT does not support to support local fragmentation.
- **Assumptions:**
  This test assumes that the data length of the icmpb.v4 traffic source exceeds the MTU for used by the DUT and Maxmwell. It also assumes that the DUT's ICMP echo server does not truncate echo replies. If none of this is the case then the results of this test are not valid.
- **Expectations:**
  The DUT must send an echo reply.
- **References:**
  RFC 1122: section 3.3.3 {Local fragmentation of outgoing packets}
- **Last Run Results:**
  PASS: icmpb.v4 got echo response, which is an expected result.

IPv4.Fragment.029 **Pass**

- **May fragment to MTU of 576 for unknown paths to off-net destinations.**
- This test attempts to verify that the MTU "MAY" limit requirement of RFC 1122, section 3.3.3 is being applied. The requirement says the DUT should only send datagrams containing 576 or fewer octets to non-local network destinations when the path MTU is unknown. The test attempts to verify this by sending fragmented datagrams smaller than 576 octets so that they appear to come from the default gateway from off-net address 74.125.224.242. The replies could be aggregated into datagrams larger than 576, but per the RFC requirement, they should not be aggregated.
- **Assumptions:**
  The default gateway must be set on the DUT and the traffic sources must create replies containing enough data in fragments that they would normally aggregate into datagrams larger than 576.
- **Expectations:**
  The DUT should not reply with any datagram larger than 576 octets. It

must sent fragmented packets if necessary.
- **References:**
  RFC 1122: section 3.3.3 {Send max 576 to off-net destination}
- **Last Run Results:**
  PASS: The DUT replied to icmp.v4 with datagrams no larger than 0 octets long; less than the RFC limit of 576.
  PASS: The DUT replied to udp.v4 with datagrams no larger than 0 octets long; less than the RFC limit of 576.

IPv4.Fragment.900 **Skipped**

- **Fragment: User set payload length. Order: User set. Overlap: User set.**
- Datagrams are split into fragments with a payload size set by the user, with an overlap set by the user, and transmitted in an order set by the user.
  The datagram will be fragmented such that each fragment contains A bytes in the payload (except the last fragment, which may be smaller). The payload of each fragment is taken from the original payload starting B bytes from where the last fragment was taken. (So if no overlap or gaps are intended then A and B should be set equal.)
  The sequence in which the fragments are transmitted and the optional generation of no-payload fragments is determined by the numerical value of C, whose bits are used as follow:

  Bits 0 to 3: A value of 0 causes the fragments to be transmitted in reverse order (that is, the fragment with the largest offset is transmitted first and the fragment with offset zero is transmitted last). A value of 1 causes fragments to be tranmitted in offset-order (that is, the fragment with offset zero is transmitted first and the fragment with the largest offset is transmitted last). Values of 2 through 15 cause intermediate fragments to be dropped.

  Bit 4: A value of 1 enables generation of an additional no-payload fragment for each fragment that has its 'more fragments' flag = 1. That is, all but the last fragment.

  Bit 5: A value of 1 enables generation of an additional no-payload fragment for each fragment having a 'more fragments' flag = 0. That is, the last fragment.

  Bit 6: A value of 0 causes the no-payload fragment to be sent before the triggering fragment. A value of 1 causes a reverse transmission order.

  Here are some common values for C (note that hexadecimal may be entered by prefixing the number with 0x):
  0: Fragments transmitted such that the offset field is in descending order.

1: Fragments transmitted such that the offset field is in ascending order.

16 (0x10): Bit 4 is set, so a no-payload fragment is also sent for all but the last data fragment. Note that descending order is used because of value in bits 0 to 3.

32 (0x20): Bit 5 is set, so a no-payload fragment is also sent for the last data fragment.

48 (0x30): Bits 4 and 5 are set so a no-payload fragment is sent for each loaded fragment, including the last.

64 (0x40): Bit 6 is set, so the no-payload fragment is sent before its corresponding loaded fragment.

114 (0x72): Bits 4, 5, and 6 are set and the integer value in bits 0 to 3 is 2 (fragments are skipped, which means reassembly will not be possible).
PARAMETERS:
A=(8,8192)
B=(8,8192)
C=(0, 1024)
General Parameter A: The number of bytes placed in the payload of each fragment.
This number ranges from 8 to 8192. General Parameter B: The beginning of each generated fragment begins this many bytes from beginning of the previously generated fragment (as measured from the original datagram). If this value is not set to a multiple of 8 then it is rounded down to the nearest multiple.
This number ranges from 8 to 8192. General Parameter C: The order in which the fragments are transmitted is specified by this number. Normally this should be set to 1 (normal order) or 0 (reverse order). Any other value causes fragments to be skipped (e.g. a value of 2 causes 1 of every 2 fragments to be dropped, a value of 3 drops 2 out of 3 fragments, and so on). There is no setting available to send fragments in reverse order and also drop any of them as there is for normal order. Please see the discussion under 'What the Test Does' for more information on the use of the higher bits of this value.
This number ranges from 0 to 1024.

- **Assumptions:**
  None.
- **Expectations:**
  Assuming no dropped fragments, the fragments must be reassembled into the original packet by the receiving IP layer and delivered to the next higher layer. If fragments are set to be dropped, then after the reassembly timeout of 60 seconds expires the reassembly must be abandoned and if the first fragment (i.e. offset zero) was received by the DUT then an ICMP

Time Exceeded message should be sent to the source host.
- **References:**
  RFC 791: section 2.3
  RFC 791: section 3.2
  RFC 1122: section 3.3.2 {Able to reassemble incoming datagrams}
  RFC 1122: section 3.2.1.4 {Support reassembly}
- **Last Run Results:**
  Skipped because no valid traffic sources specified.

IPv4.Framing.000 **Pass**

- **Test to insure the DUT is in a sane state.**
- This is a 'sanity' check. During multiple automated test runs the impairments may cause the DUT stack or echo servers to enter unresponsive states. This test passes traffic through unimpaired so that if the DUT has entered an unresponsive state one or more failures responses will be returned, indicating one of the tests since the last successful sanity check caused the DUT to fail. The tests since that last successful sanity check will need to be manually analyzed to determine the test that caused the DUT to enter the 'bad' state.
- **Assumptions:**
  The DUT should be running the echo servers or other applications that are expected to reply to Maxwell generated traffic for the desired set of tests.
- **Expectations:**
  The DUT is expected to reply to the traffic sources with the responses defined for them in the XML sources files.
- **References:**
  None applicable.
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.
  PASS: icmp.v6 got echo response, which is an expected result.
  PASS: icmpb.v6 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: udpb.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.
  PASS: tcp.v6 got echo response, which is an expected result.

IPv4.Framing.001 **Pass**

- **MAC frame size extended to 1500 octet payload size.**
- IP datagram that arrives that is contained in a MAC frame smaller than a specified size will be re-wrapped into a MAC frame of that specified size. This has the effect of making small IP datagrams occupy only a small portion of the MAC frame.
  In this test all IP datagrams are extended, if possible so that they have a

layer 2 length of 1500 plus the MAC header, i.e. from the point of view of the MAC frame the payload is 1500 octets.
This exercises the IP stack's ability to properly obtain IP length information from the IP header rather than from the enclosing layer 2 frame.
This test also verifies (rather redundantly, but for conformance completion) that the DUT is using ARP and RFC 894 Ethernet ecapsulation.

- **Assumptions:**
  MTU of the DUT is at or above 1500 (the default for Ethernet).
- **Expectations:**
  The DUT must use the IP length field and not the MAC frame length to determine the size of the IP datagram. The DUT should react to the IP packet flow in the same way they would were the frame padding to be absent.
- **References:**
  RFC 1122: section 2.3.3 {Send & receive RFC-894 encapsulation}
  RFC 1122: section 2.3.3 {Use ARP on Ethernet and IEEE 802 nets}
  RFC 791
  RFC 894
  RFC 1042
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Framing.002 **Skipped**

- **MAC frame size extended to 2048 octet payload size.**
- IP datagram that arrives that is contained in a MAC frame smaller than a specified size will be re-wrapped into a MAC frame of that specified size. This has the effect of making small IP datagrams occupy only a small portion of the MAC frame.
  In this test all IP datagrams are extended, if possible so that they have a layer 2 length of 2048 plus the MAC header, i.e. from the point of view of the MAC frame the payload is 2048 octets. With a normal 14 octet MAC header, this gives a total length (excluding CRC) of 2062 octets. Packets with 802 frames or VLAN/QoS tags will have somewhat longer headers.
  This exercises the IP stack's ability to properly obtain IP length information from the IP header rather than from the enclosing layer 2 frame.
- **Assumptions:**
  The DUT is assumed to support Jumbograms/Jumbo Frames. This test requires that the MTU of the interfaces on Maxwell and the DUT be increased. This is done manually. The command to do this to an interface is:
  ifconfig eth# mtu XXXXX
  where:
  # is the number of the the interface (remember that both interfaces need

to be adjusted).
XXXXX is the desired MTU. For the Intel NICs used in Maxwell an appropriate value would be 11000 (There is no harm if the MTU set by ifconfig is higher than what the test requires.)

- **Expectations:**
  The DUT must use the IP length field and not the MAC frame length to determine the size of the IP datagram. The DUT should react to the IP packet flow in the same way they would were the frame padding to be absent.
- **References:**
  RFC 791
  RFC 894
  RFC 1042
  RFC 2675
- **Last Run Results:**
  SKIPPED: icmp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)
  SKIPPED: udp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)
  SKIPPED: tcp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)

IPv4.Framing.003 **Skipped**

- **MAC frame size extended to 4096 octet payload size.**
- IP datagram that arrives that is contained in a MAC frame smaller than a specified size will be re-wrapped into a MAC frame of that specified size. This has the effect of making small IP datagrams occupy only a small portion of the MAC frame.
  In this test all IP datagrams are extended, if possible so that they have a layer 2 length of 4096 plus the MAC header, i.e. from the point of view of the MAC frame the payload is 4096 octets. With a normal 14 octet MAC header, this gives a total length (excluding CRC) of 4110 octets. Packets with 802 frames or VLAN/QoS tags will have somewhat longer headers. This exercises the IP stack's ability to properly obtain IP length information from the IP header rather than from the enclosing layer 2 frame.
- **Assumptions:**
  The DUT is assumed to support Jumbograms/Jumbo Frames. This test requires that the MTU of the interfaces on Maxwell and the DUT be increased. This is done manually. The command to do this to an interface is:
  ifconfig eth# mtu XXXXX
  where:

is the number of the the interface (remember that both interfaces need to be adjusted).
XXXXX is the desired MTU. For the Intel NICs used in Maxwell an appropriate value would be 11000 (There is no harm if the MTU set by ifconfig is higher than what the test requires.)

- **Expectations:**
The DUT must use the IP length field and not the MAC frame length to determine the size of the IP datagram. The DUT should react to the IP packet flow in the same way they would were the frame padding to be absent.
- **References:**
RFC 791
RFC 894
RFC 1042
RFC 2675
- **Last Run Results:**
SKIPPED: icmp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)
SKIPPED: udp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)
SKIPPED: tcp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)

IPv4.Framing.004 **Skipped**

- **MAC frame size extended to 8192 octet payload size.**
- IP datagram that arrives that is contained in a MAC frame smaller than a specified size will be re-wrapped into a MAC frame of that specified size. This has the effect of making small IP datagrams occupy only a small portion of the MAC frame.
In this test all IP datagrams are extended, if possible so that they have a layer 2 length of 8192 plus the MAC header, i.e. from the point of view of the MAC frame the payload is 8192 octets. With a normal 14 octet MAC header, this gives a total length (excluding CRC) of 8206 octets. Packets with 802 frames or VLAN/QoS tags will have somewhat longer headers. This exercises the IP stack's ability to properly obtain IP length information from the IP header rather than from the enclosing layer 2 frame.
- **Assumptions:**
The DUT is assumed to support Jumbograms/Jumbo Frames. This test requires that the MTU of the interfaces on Maxwell and the DUT be increased. This is done manually. The command to do this to an interface is:
ifconfig eth# mtu XXXXX

where:
is the number of the the interface (remember that both interfaces need to be adjusted).
XXXXX is the desired MTU. For the Intel NICs used in Maxwell an appropriate value would be 11000 (There is no harm if the MTU set by ifconfig is higher than what the test requires.)

- **Expectations:**
The DUT must use the IP length field and not the MAC frame length to determine the size of the IP datagram. The DUT should react to the IP packet flow in the same way they would were the frame padding to be absent.
- **References:**
RFC 791
RFC 894
RFC 1042
RFC 2675
- **Last Run Results:**
SKIPPED: icmp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)
SKIPPED: udp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)
SKIPPED: tcp.v4: Packets exchanged with the DUT did not match those needed to either perform or grade the test. (Traffic source reports: echo response.)

IPv4.Framing.005 **Fail**

- **Receive RFC-1042 encapsulation.**
- IEEE 802 encapsulation (RFC 1042) is used for the ICMP echo request instead of Ethernet encapsulation (RFC 894). The DUT is expected to respond with the same encapsulation.
- **Assumptions:**
None.
- **Expectations:**
The DUT should reply to the echo request. Any reply must use RFC 1042 encapsulation.
- **References:**
RFC 1122: section 2.3.3 {Receive RFC-1042 encapsulation}
RFC 1122: section 2.3.3 {Send K1=6 encapsulation}
- **Last Run Results:**
FAIL: icmp.v4 returned "No echo response", which is not an expected result.

IPv4.Options.000 **Pass**

- **Test to insure the DUT is in a sane state.**
- This is a 'sanity' check. During multiple automated test runs the impairments may cause the DUT stack or echo servers to enter unresponsive states. This test passes traffic through unimpaired so that if the DUT has entered an unresponsive state one or more failures responses will be returned, indicating one of the tests since the last successful sanity check caused the DUT to fail. The tests since that last successful sanity check will need to be manually analyzed to determine the test that caused the DUT to enter the 'bad' state.
- **Assumptions:**
  The DUT should be running the echo servers or other applications that are expected to reply to Maxwell generated traffic for the desired set of tests.
- **Expectations:**
  The DUT is expected to reply to the traffic sources with the responses defined for them in the XML sources files.
- **References:**
  None applicable.
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.
  PASS: icmp.v6 got echo response, which is an expected result.
  PASS: icmpb.v6 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: udpb.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.
  PASS: tcp.v6 got echo response, which is an expected result.

IPv4.Options.001 **Pass**

- **Append one End of Options List (EOOL) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding.
  In this test the option is of type 0 - End of Options List.
- **Assumptions:**
  None.
- **Expectations:**
  The packet should be processed by the IP layer in the same manner as if the appended options were not there.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

PASS: udp.v4 got echo response, which is an expected result.
PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.002 **Pass**

- **Append one No Operation (NOP) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding. In this test the option is of type 1 - No Operation.
- **Assumptions:**
  None.
- **Expectations:**
  The packet should be processed by the IP layer in the same manner as if the appended options were not there.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.003 **Pass**

- **Append one Security (SEC) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding. In this test the option is of type 2 - Basic Security. The option that will be generated has a length of 3, a classification level of *unclassified* and no Protection Authority Flags.
- **Assumptions:**
  None.
- **Expectations:**
  This option is defined inconsistently in two different RFCs: RFC791 and RFC2402 (indirectly via RFC1108). The IANA website references the RFC1108 definition on http://www.iana.org/assignments/ip-parameters This code uses the RFC1108 definition. Ethereal uses the RFC791 definition. RFC1122 indicates that the security options described in RFC791 are obsolete.
  The expected outcome for recent stacks is that the option may be ignored.
- **References:**
  RFC 791: section 3.1

RFC 1122: section 3.2.1.8a {Security option}

- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.004 **Fail**

- **Append one Loose Source Route (LSR) option.**
- If space is available in the IPv4 header, an option of type 131 - Loose Source Route (LSR) is added. The original source and destination addreses are included in the route and the length is set to 11. The pointer is set to 12, indicating the routing has completed. The inclusion of the original source address in the route is intended to mimic the common (incorrect) interpretation discussed in RFC 1122, section 3.2.1.8c.
- **Assumptions:**
  The host is assumed to support source routing; otherwise the results of this test are not relevant.
- **Expectations:**
  Since all hosts are permitted to perform local source-routing (i.e. next hop is on the same physical interface) without restriction, the receiving host may forward the packet to itself. The packet must then be delivered to the next higher layer and eventually a response sent.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8c {Originate & terminate Source Route options}
  RFC 1122: section 3.2.1.8c {Datagram with completed SR passed up to TL}
  RFC 1122: section 3.2.1.8c {Build correct (non-redundant) return route}
  RFC 1122: section 3.2.1.8c {Send multiple SR options in one header}
  RFC 1122: section 3.2.2.6 {Reverse and reflect Source Route option}
- **Last Run Results:**
  FAIL: icmp.v4 got ICMPv4 Destination Unreachable (type 3, code 5), which is not an expected result.
  FAIL: udp.v4 got no echo response, which is not an expected result.
  FAIL: tcp.v4 got operation not supported on connect call, which is not an expected result.

IPv4.Options.005 **Pass**

- **Append one Time Stamp (TS) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding. In this test the option is of type 4 - Time Stamp. The

option that will be generated has a length of 20. The timestamp option will have two timestamps each with an IP address. The first IP address is 10.20.30.40 and the timestamp is roughly 25 milliseconds before *now* The second IP address is zero and has zero for a timestamp. The pointer field is set to point at that second timestamp area. This means that the next hop should fill in the address and timestamp.

- **Assumptions:**
  None.
- **Expectations:**
  The destination host must add the current timestamp to the Timestamp option before passing the option and packet to the next higher layer.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8e {Timestamp option}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.006 **Pass**

- **Append one Extended Security (E-SEC) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding. In this test the option is of type 5 - Extended Security. This option always is coupled with the Basic Security option. (However the relative order of those options is not specified.) There is an issue with the security option regarding which RFC governs its definition, RFC791 or RFC1108. This test uses the latter. The option that will be generated has a length of 6.
- **Assumptions:**
  None.
- **Expectations:**
  This option is defined inconsistently in two different RFCs: RFC791 and RFC2402 (indirectly via RFC1108). The IANA website references the RFC1108 definition on http://www.iana.org/assignments/ip-parameters This code uses the RFC1108 definition. Ethereal uses the RFC791 definition. RFC1122 indicates that the security options described in RFC791 are obsolete.
  The expected outcome for recent stacks is that the option may be ignored.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8a {Security option}
- **Last Run Results:**

PASS: icmp.v4 got echo response, which is an expected result.
PASS: udp.v4 got echo response, which is an expected result.
PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.007 **Pass**

- **Append one Record Route (RR) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding.
  In this test the option is of type 7 - Record Route.
  The option that will be generated has a length of 11.
  The option that is generated will have space for two IP addresses.
  The first will contain 10.9.8.7 and the second 192.168.10.9.
  The pointer is set to a position such that the second address would be the next to be written.
- **Assumptions:**
  None.
- **Expectations:**
  The destination host may record its address into the open position and advance the pointer. It then should deliver the packet to the next higher layer.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8d {Record Route option}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.008 **Pass**

- **Append one Stream ID (SID) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding.
  In this test the option is of type 8 - Stream ID.
  The option that will be generated has a length of 4.
  The stream ID itself will have a value of 0.
- **Assumptions:**
  None.
- **Expectations:**

Per RFC 1122 this option is obsolete and should be silently ignored during processing of the packet. If the DUT responds with a Stream ID option the grader will detect that and return a failing grade.

- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8b {Silently ignore Stream Identifer option}
  RFC 1122: section 3.2.1.8b {Send Stream Identifier option}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.009 **Fail**

- **Append one Strict Source Route (SSR) option.**
- If space is available in the IPv4 header, an option of type 137 - Strict Source Route (SSR) is added. The original source and destination addreses are included in the route and the length is set to 11. The pointer is set to 12, indicating the routing has completed. The inclusion of the original source address in the route is intended to mimic the common (incorrect) interpretation discussed in RFC 1122, section 3.2.1.8c.
- **Assumptions:**
  The host is assumed to support source routing; otherwise the results of this test are not relevant.
- **Expectations:**
  Since all hosts are permitted to perform local source-routing (i.e. next hop is on the same physical interface) without restriction, the receiving host may forward the packet to itself. The packet must then be delivered to the next higher layer and eventually a response sent.
- **References:**
  RFC 791: section 3.1
  RFC 1122: section 3.2.1.8c {Originate & terminate Source Route options}
  RFC 1122: section 3.2.1.8c {Datagram with completed SR passed up to TL}
  RFC 1122: section 3.2.1.8c {Build correct (non-redundant) return route}
  RFC 1122: section 3.2.1.8c {Send multiple SR options in one header}
  RFC 1122: section 3.2.2.6 {Reverse and reflect Source Route option}
- **Last Run Results:**
  FAIL: icmp.v4 got ICMPv4 Destination Unreachable (type 3, code 5), which is not an expected result.
  FAIL: udp.v4 got no echo response, which is not an expected result.
  FAIL: tcp.v4 got operation not supported on connect call, which is not an expected result.

IPv4.Options.010 **Pass**

- **Append one Router Alert (RTRALT) option.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation. The options area of the IP header is filled out with 0 through 3 bytes of zero that serve the dual purpose of indicating end-of-options and padding.
  In this test the option is of type 20 - Router Alert.
  The option that will be generated has a length of 4.
  The value octets of the option will be zero.
- **Assumptions:**
  None.
- **Expectations:**
  This option should be ignored by hosts and may be ignored by routers that do not support protocols requiring its use.
- **References:**
  RFC 2113
  RFC 1122: section 3.2.1.8 {IP layer silently ignore unknown options}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.

IPv4.Options.011 **Pass**

- **Append an unknown option with a length of 0.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, however the length octet has a value of zero. The option type is 31 which has not been allocated by IANA.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
  Notes:
  Some IP stacks in the past have gone into infinite loops when they try to process options with zero length fields.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is

an expected result.
PASS: udp.v4 got no echo response, which is an expected result.
PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.012 **Pass**

- **Append an unknown option with a length of 1.**
- This test adds an unknown option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, however the length octet has a value of 1. The option type is 31 which has not been allocated by IANA.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.013 **Pass**

- **Append an unknown option with a length that overflows the header by 1.**
- This test adds an unknown option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, meaning an IP packet with a header length of 28 octets. However the length octet in the option has a value of 9, which is too large to fit into the IP header of this packet. The option type is 31 which has not been allocated by IANA.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an

ICMP parameter problem message.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.014 **Pass**

- **Append an unknown option with a claimed length of 41.**
- This test adds an unknown option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, meaning an IP packet with a header length of 28 octets. However the length octet in the option has a value of 41, which is too large to fit into the IP header of any IPv4 packet.
  The option type is 31 which has not been allocated by IANA.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.015 **Pass**

- **Append a RTRALT option with a length of 0.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause

fragmentation.
The option to be added here consists of 6 octets, however the length octet has a value of zero.
The option type is 20 (148 with the copy/class bits), Router Alert. This option is defined to have a fixed size of 4.

- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
  Notes:
  Some IP stacks in the past have gone into infinite loops when they try to process options with zero length fields.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.016 **Pass**

- **Append a RTRALT option with a length of 1.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, however the length octet has a value of 1.
  The option type is 20 (148 with the copy/class bits), Router Alert. This option is defined to have a fixed size of 4.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**

PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
PASS: udp.v4 got no echo response, which is an expected result.
PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.017 **Pass**

- **Append a RTRALT option with a length that overflows the header by 1.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, meaning an IP packet with a header length of 28 octets. However the length octet in the option has a value of 9, which is too large to fit into the IP header of this packet.
  The option type is 20 (148 with the copy/class bits), Router Alert. This option is defined to have a fixed size of 4.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.018 **Pass**

- **Append a RTRALT option with a claimed length of 41.**
- This test adds an option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added here consists of 6 octets, meaning an IP packet with a header length of 28 octets. However the length octet in the option has a value of 41, which is too large to fit into the IP header of any IPv4 packet.
  The option type is 20 (148 with the copy/class bits), Router Alert. This

option is defined to have a fixed size of 4.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT should notice the bad length. The DUT should respond with an ICMP parameter problem message.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.2 {Included octets same as received}
  RFC 1122: section 3.2.2.5 {Send Parameter Problem messages}
- **Last Run Results:**
  PASS: icmp.v4 got ICMPv4 Parameter Problem (type 12, code 0), which is an expected result.
  PASS: udp.v4 got no echo response, which is an expected result.
  PASS: tcp.v4 got protocol error on connect call, which is an expected result.

IPv4.Options.019 **Pass**

- **Append an unknown option with a length of 4.**
- This test adds an unknown option to IP packets unless there is already an option in the packet or the packet is so large that adding an option would cause fragmentation.
  The option to be added has a type value of 31, which has not been allocated by IANA. The length field is 4 and two zero bytes follow.
- **Assumptions:**
  None.
- **Expectations:**
  The IP and any upper layers must ignore the unknown option.
- **References:**
  RFC 791: section 3.1
  RFC 792
  RFC 1122: section 3.2.1.8 {IP layer silently ignore unknown options}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.

IPv4.ICMP.000 **Pass**

- **Test to insure the DUT is in a sane state.**
- This is a 'sanity' check. During multiple automated test runs the impairments may cause the DUT stack or echo servers to enter unresponsive states. This test passes traffic through unimpaired so that if the DUT has entered an unresponsive state one or more failures responses will be returned, indicating one of the tests since the last successful sanity

check caused the DUT to fail. The tests since that last successful sanity check will need to be manually analyzed to determine the test that caused the DUT to enter the 'bad' state.

- **Assumptions:**
  The DUT should be running the echo servers or other applications that are expected to reply to Maxwell generated traffic for the desired set of tests.
- **Expectations:**
  The DUT is expected to reply to the traffic sources with the responses defined for them in the XML sources files.
- **References:**
  None applicable.
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.
  PASS: icmpb.v4 got echo response, which is an expected result.
  PASS: icmp.v6 got echo response, which is an expected result.
  PASS: icmpb.v6 got echo response, which is an expected result.
  PASS: udp.v4 got echo response, which is an expected result.
  PASS: udpb.v4 got echo response, which is an expected result.
  PASS: tcp.v4 got echo response, which is an expected result.
  PASS: tcp.v6 got echo response, which is an expected result.

IPv4.ICMP.001 **Pass**

- **Implements IPV4 ICMP echo server and needs no gateway.**
- All other tests that send traffic to the DUT ICMP echo server assume it is operational. Passing this test allows you to claim conformance to at least the ICMP echo server requirement of RFC 1122.
- **Assumptions:**
  The DUT must be directly connected to Maxwell. No intervening router can exist.
- **Expectations:**
  An echo response is expected that contains the exact same payload as sent to the DUT.
- **References:**
  RFC 1122: section 3.2.2.6 {Echo server and Echo client}
  RFC 1122: section 3.2.2.6 {Use specific-dest addr as Echo Reply src}
  RFC 1122: section 3.2.2.6 {Send same data in Echo Reply}
  RFC 1122: section 3.3.1.1 {Operate with no gateways on conn network}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.ICMP.002 **Pass**

- **Sets the code field to an unassigned value.**
- The code field is set to the value 42, which has not yet been allocated by the IANA for any of the assigned code fields.

- **Assumptions:**
  None.
- **Expectations:**
  Per the Robustness Principle, the message should be passed to the upper layer protocol. While a drop may be proper for some message types, the code field appears to be unused for echo requests and therefore an echo reply should be returned.
- **References:**
  RFC 792: page 12
  RFC 1122: section 1.2.2
  RFC 1122: section 3.2.2.6 {Send same data in Echo Reply}}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.ICMP.003 **Pass**

- **Sets the checksum to zero.**
- The sets the checksum field to zero.
- **Assumptions:**
  None.
- **Expectations:**
  The packet must be silently dropped. Unlike the UDP checksum field, the ICMP checksum field computation is not optional.
- **References:**
  RFC 792
  RFC 1122: section 3.2.2
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.

IPv4.ICMP.004 **Pass**

- **Adjust message so checksum is -0 and set checksum field to -0.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  Bytes 9 and 10 of the ICMP message are adjusted so that the checksum yields -0. (Because it is expected that there will always be at least one non-zero value in the message, the checksum can never be +0.) Since it is the complement of the checksum (not the checksum itself!) that is stored in the checksum field, the checksum field can therefore never contain all ones (see discussion section of RFC 1624.) But instead of +0 in the checksum field, -0 is stored. However, the ones complement sum check must still yield all ones, indicating no transmission errors. NOTE: This test was previously named IPv4.ICMP.016
- **Assumptions:**
  The ICMP message must contain at least 10 bytes and the 9th and 10th

bytes of the ICMP message are assumed to be malleable; that is, the ultimate destination is expected to ignore their contents since they need to be adjusted to satisfy the requirements of the impairment.
- **Expectations:**
  The DUT must accept the ICMPv4 message.
- **References:**
  RFC 792
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.ICMP.005 **Pass**

- **Adjust message so checksum is -0 and set checksum field to +0.**
- NOTE: The contents of the checksum field in the header and the checksum are different - the checksum field holds the ones complement of the checksum.
  Bytes 9 and 10 of the ICMP message are adjusted so that the checksum yields -0. (Because it is expected that there will always be at least one non-zero value in the message, the checksum can never be +0.) Since it is the complement of the checksum (not the checksum itself!) that is stored in the checksum field, the checksum field can therefore never contain all ones (see discussion section of RFC 1624.)
- **Assumptions:**
  The ICMP message must contain at least 10 bytes and the 9th and 10th bytes of the ICMP message are assumed to be malleable; that is, the ultimate destination is expected to ignore their contents since they need to be adjusted to satisfy the requirements of the impairment. NOTE: This test was previously named IPv4.ICMP.017
- **Expectations:**
  The DUT must accept the ICMPv4 message.
- **References:**
  RFC 792
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.ICMP.006 **Pass**

- **Adjust message so checksum is +1 and set checksum field to +0.**
- NOTE: The contents of the checksum field in the header and the checksum are different - the checksum field holds the ones complement of the checksum.
  The 9th and 10th message bytes are adjusted so that the checksum yields +1. However, the checksum field is set to +0. NOTE: This test was previously named IPv4.ICMP.018

- **Assumptions:**
  The ICMP message must contain at least 10 bytes and the 9th and 10th bytes of the ICMP message are assumed to be malleable; that is, the ultimate destination is expected to ignore their contents since they need to be adjusted to satisfy the requirements of the impairment.
- **Expectations:**
  The DUT should silently reject the ICMPv4 message.
- **References:**
  RFC 792
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.

IPv4.ICMP.007 **Pass**

- **Set checksum field so it contains -1.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  The checksum field is set to -1 (0xFFFE in hex) but the message is not adjusted to yield a checksum of +1. NOTE: This test was previously named IPv4.ICMP.019
- **Assumptions:**
  Assumes the original checksum field does not contain -1. If it does then any errors reported by the automated test are incorrect.
- **Expectations:**
  The DUT should silently reject the ICMPv4 message.
- **References:**
  RFC 792
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.

IPv4.ICMP.008 **Pass**

- **Invert the checksum field bit values.**
- NOTE: The contents of the checksum field in the header and the checksum itself are different - the checksum field holds the ones complement of the checksum.
  The checksum field bits are all inverted. NOTE: This test was previously named IPv4.ICMP.020
- **Assumptions:**
  Assumes the original checksum field does not contain +/-0. If it does then any errors reported by the automated test are incorrect.
- **Expectations:**
  The DUT should silently reject the ICMPv4 message.

- **References:**
  RFC 792
  RFC 1624: section 3
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.

IPv4.ICMP.009 **Pass**

- **Sets type field to an unassigned value.**
- The type field is set to the value 42, which has not yet been allocated by the IANA. NOTE: This test was previously named IPv4.ICMP.001.
- **Assumptions:**
  None.
- **Expectations:**
  When an ICMP message of unknown type is received, it must be silently discarded.
- **References:**
  RFC 792
  RFC 1122: section 3.2.2 {Silently discard ICMP msg with unknown type}
- **Last Run Results:**
  PASS: icmp.v4 got no echo response, which is an expected result.

IPv4.ICMP.010 **Pass**

- **Force UDP port unreachable condition with link-layer broadcast.**
- The destination port of a UDP/IPv4 datagram is changed to 65534 and the link-layer address is changed to the broadcast address FF:FF:FF:FF:FF:FF. The datagram's destination is the DUT and no service is listening on that port.
- **Assumptions:**
  None.
- **Expectations:**
  The DUT must not send an error response in response to any packet containg a link-layer broadcast.
- **References:**
  RFC 1122: section 3.2.2 {Link-layer b'cast}
- **Last Run Results:**
  PASS: udp.v4 got no echo response, which is an expected result.

IPv4.ICMP.011 **Pass**

- **Destination unreachable serves only as a hint.**
- This test sends a host destination unreachable message to the DUT. It verifies that the unreachable message does not cause subsequent packets to not be sent to Maxwell. It does this by sending an ICMP echo request and when it receives the reply it drops it and issues the host destination

unreachable message. It then sends a second echo request and expects a reply from the second one.
This test is not a completely proper test because the existence of the second echo request might be used by the DUT to consider the host available again after it had flagged it as unreachable. A full and proper test can only be done in a two step process with the DUT initiating TCP or UDP connections. This is outside the capability of this test system.

- **Assumptions:**
  This test has assumed that the DUT is not using the positive advice mechanism mentioned in RFC 1122: "Packets arriving from a particular link-layer address are evidence that the system at this address is alive."
- **Expectations:**
  The DUT must respond to the second echo requests
- **References:**
  RFC 1122: section 3.2.2.1 {Interpret Dest Unreach as only hint}
- **Last Run Results:**
  PASS: icmp.v4 got echo response, which is an expected result.

IPv4.ICMP.012 **Pass**

- **Verify silent discard of bad redirects.**
- This test checks that bad ICMPv4 redirects are discarded. It does this by forcing the DUT to reply to an ICMP echo request via its default gateway. On getting the DUT's first echo reply, an ICMP NETWORK REDIRECT is sent that changes the gateway to remote IP address 242.224.125.74 and an alternate MAC address. If the DUT sends an ARP request on the 242.224.125.74 address then the DUT fails since the DUT obviously did not silently discard the bad redirect. The test is run twice, using host and network redirects.
- **Assumptions:**
  The IPv4 gateway field must be set so that the maxtap inteface acts as a gateway. The DUT must also be configured to accept redirects. Some systems have redirects disabled by default as a security precaution.
- **Expectations:**
  The DUT should silently discard the bad redirect request.
- **References:**
  RFC 1122: section 3.2.2.2 {Discard illegal Redirect}
- **Last Run Results:**
  PASS: DUT silently dropped the bad redirect.

IPv4.ICMP.013 **Pass**

- **Verify record route and time stamp option operation.**
- Both a Record Route and Time Stamp option are added to the IPv4 header of ICMPv4 Echo Request messages.
- **Assumptions:**

None.
- **Expectations:**
  The DUT should include in its Echo Response an IPv4 Record Route option containing the DUT's IPv4 address and a Time Stamp option with updated time information. The test will only be graded as passing if both conditions are true.
- **References:**
  RFC 1122: section 3.2.2.6 {Reflect Record Route, Time Stamp options}
- **Last Run Results:**
  PASS: Received well-formed IPv4 Record Route and Time Stamp options.

IPv4.ICMP.900 **Skipped**

- **At a user specified location, set a user specified byte value.**
- The user may use this test to specify a byte offset from the beginning of the ICMP messages at which a user specified byte value should be overwritten. The byte offset must be entered into General Parameter A and the byte value must be entered into General Parameter B.
  PARAMETERS:
  A=(0,1024)
  B=(0,255)
  General Parameter A: The byte location where the General Parameter B value should written.
  This number ranges from 0 to 1024. General Parameter B: The byte value to overwrite.
  This number ranges from 0 to 255.
- **Assumptions:**
  None.
- **Expectations:**
  The expected outcome cannot be specified for this user-adjustable test.
- **References:**
  RFC 792
  RFC 1122: section 3.2.2
- **Last Run Results:**
  Skipped because no valid traffic sources specified.

IPv4.ICMP.901 **Skipped**

- **At a user specified location, set a user specified short word value.**
- The user may use this test to specify a byte offset from the beginning of the ICMP messages at which a user specified short word (two byte) value should be overwritten. The byte offset must be entered into General Parameter A and the short word value must be entered into General Parameter B.
  PARAMETERS:
  A=(0,1024)