# Network Protocols TCP, IP and Configuration

## EFI\_TCP4\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_TCP4\_PROTOCOL Section.

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.1.1 | 0xf7c924b2, 0xaaa6, 0x4729, 0xb1, 0xd0, 0x71, 0xf8, 0xed, 0xc8, 0x81, 0x8f | EFI\_TCP4\_PROTOCOL.GetModeData() – invokes GetModeData() with a Tcp4State value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with a Tcp4Statevalue of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.2 | 0xd39219b6, 0xa262, 0x4797, 0xac, 0x44, 0x35, 0xe5, 0x46, 0xc0, 0xe9, 0xc8 | EFI\_TCP4\_PROTOCOL.GetModeData() – invokes GetModeData() with a Tcp4ConfigData value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with a Tcp4ConfigData value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.3 | 0x7be1ddb5, 0xf3bf, 0x4eb3, 0x87, 0x52, 0x9a, 0xf6, 0x91, 0x6c, 0x51, 0xc5 | EFI\_TCP4\_PROTOCOL.GetModeData() – invokes GetModeData() with a Ip4ModeData value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with a Ip4ModeData value ofNULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.4 | 0x6255190b, 0x3eb5, 0x40e9, 0xbd, 0x24, 0x26, 0x85, 0xfc, 0x87, 0xab, 0x29 | EFI\_TCP4\_PROTOCOL.GetModeData() – invokes GetModeData() with a *MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with a *MnpConfigData* value ofNULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.5 | 0x62f96356, 0x53d3, 0x4fdd, 0xb1, 0x36, 0x12, 0x53, 0xc2, 0xb0, 0x14, 0x8e | EFI\_TCP4\_PROTOCOL.GetModeData() – invokes GetModeData() with a *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3.Call EFI\_TCP4\_PROTOCOL.GetModeData() with a *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.6 | 0xf753264f, 0x22d0, 0x4e19, 0x81, 0x81, 0xf3, 0x4d, 0xd9, 0xf6, 0xdb, 0x59 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4State andTcp4ConfigDatavalue of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3.Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4State andTcp4ConfigDatavalue ofNULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.7 | 0x0848d02d, 0x3463, 0x4f06, 0xb1, 0x6e, 0xce, 0xd1, 0x32, 0x3b, 0x53, 0xd2 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4StateandIp4ModeDatavalue of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3.Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4Stateand Ip4ModeDatavalue ofNULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.8 | 0xa92b1577, 0x6d14, 0x4d77, 0x9f, 0x5b, 0x85, 0xba, 0x55, 0xf8, 0x1d, 0x52 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4State and*MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3.Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4State and*MnpConfigData*value ofNULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.9 | 0x31388819, 0x2579, 0x414e, 0x89, 0x0f, 0xfe, 0xc9, 0xbe, 0x08, 0x8c, 0x37 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4State and*SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4State and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.10 | 0xec2502c3, 0xdf73, 0x4bff, 0xa4, 0xac, 0xaf, 0x5e, 0x77, 0x3d, 0xbf, 0xa1 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4ConfigData andIp4ModeData value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4ConfigData and Ip4ModeData value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.11 | 0x32100ad2, 0xbc14, 0x426b, 0x86, 0xee, 0x0e, 0xc1, 0x8e, 0xb3, 0x11, 0xb2 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4ConfigData and*MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4ConfigData and *MnpConfigData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.12 | 0x3ae2f864, 0x8963, 0x48ca, 0xbc, 0xa5, 0x01, 0x0d, 0xdf, 0x13, 0x9e, 0xb1 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Tcp4ConfigData and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Tcp4ConfigData and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.13 | 0xc72c71bf, 0x781f, 0x4a08, 0xac, 0xa1, 0xb0, 0x1f, 0xbc, 0x79, 0x91, 0x60 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Ip4ModeData and *MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Ip4ModeData and *MnpConfigData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.14 | 0x86fb248c, 0x3238, 0x411e, 0xa6, 0xa5, 0x41, 0x1c, 0x21, 0x42, 0x82, 0xc4 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the Ip4ModeData and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the Ip4ModeData and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.15 | 0xdddaf809, 0xa972, 0x4376, 0xb2, 0xdb, 0x1a, 0x35, 0x14, 0xcc, 0x88, 0x0a | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with both the *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with both the *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.16 | 0xf6873b19, 0xbdef, 0x4bac, 0x93, 0x4d, 0x55, 0xe0, 0x87, 0x06, 0x67, 0x2e | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Tcp4ConfigData and Ip4ModeData value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Tcp4ConfigData and Ip4ModeData value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.17 | 0x8b5d7aa1, 0x9838, 0x4b5a, 0x88, 0x37, 0xa7, 0xd1, 0x93, 0x5f, 0x8e, 0x46 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Tcp4ConfigData and *MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Tcp4ConfigData and *MnpConfigData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.18 | 0x064d8786,0x876c, 0x46a2, 0x84, 0xa7, 0x1a, 0x69, 0x8a, 0x59, 0x65, 0xb0 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Tcp4ConfigData and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Tcp4ConfigData and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.19 | 0xb98bb8a0, 0xf8bd, 0x405d, 0x99, 0x6c, 0x52, 0x47, 0x3c, 0x20, 0x43, 0x38 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Ip4ModeData and *MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData()GetModeData() with the Tcp4State, Ip4ModeData and *MnpConfigData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.20 | 0x23fa07b0, 0xcd96, 0x490b, 0xa6, 0xf6, 0xe6, 0x5d, 0x8d, 0x89, 0x28, 0xc6 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Ip4ModeData and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Ip4ModeData and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.21 | 0xbfa282e9, 0x6393, 0x428f, 0x8f, 0xe1, 0x6d, 0xf2, 0xca, 0xfc, 0x9b, 0x84 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.22 | 0x245ea469, 0x0422, 0x45fa, 0x97, 0x4b, 0x0b, 0x45, 0xc2, 0xf8, 0x70, 0x27 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4ConfigData, Ip4ModeData and *MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4ConfigData, Ip4ModeData and *MnpConfigData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.23 | 0x70445b77, 0x59ec, 0x4fd1, 0xba, 0x2b, 0x9a, 0xcd, 0x7e, 0x0f, 0x78, 0x83 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4ConfigData, Ip4ModeData and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4ConfigData, Ip4ModeData and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.24 | 0xfa72381d, 0x5c30, 0x4dd1, 0xba, 0xf4, 0xff, 0xca, 0x30, 0x0a, 0x2f, 0x15 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4ConfigData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4ConfigData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.25 | 0xad6d2b6f, 0x8e2f, 0x49ed, 0xa1, 0xd8, 0x3b, 0x33, 0x69, 0x04, 0x2c, 0x2e | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.26 | 0x7d6ef330, 0x3522, 0x434d, 0x9f, 0xf7, 0x34, 0x84, 0xe4, 0x0d, 0x1f, 0xc5 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4ConfigData,Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4ConfigData,Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.27 | 0x1f83096c, 0x6342, 0x4f1a, 0xa1, 0x22, 0xe3, 0x1e, 0xd5, 0x63, 0x36, 0x53 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State,Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State,Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.28 | 0xe7f67d55, 0x5bb8, 0x400c, 0x99, 0xfc, 0x53, 0x0e, 0x5d, 0xc0, 0x1f, 0x51 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Tcp4ConfigData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Tcp4ConfigData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.29 | 0xa72e1aec, 0x5502, 0x434c, 0xb8, 0xed, 0x68, 0x0b, 0x54, 0xb2, 0xa8, 0x8e | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Tcp4ConfigData, Ip4ModeData and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Tcp4ConfigData, Ip4ModeData and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.30 | 0x59e6caf6, 0x0db0, 0x45f9, 0x91, 0x50, 0xca, 0xdb, 0x1c, 0xae, 0x9b, 0xc2 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with the Tcp4State, Tcp4ConfigData, Ip4ModeData and *MnpConfigData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with the Tcp4State, Tcp4ConfigData, Ip4ModeData and *MnpConfigData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.31 | 0x3fd1ebb6, 0x3edd, 0x4a61, 0x98, 0x8e, 0xfc, 0x92, 0xbd, 0xef, 0x8d, 0xf0 | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with all the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with all the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.32 | 0x53417686, 0xcf3b, 0x4dc5, 0x9d, 0x7b, 0x83, 0xad, 0x7c, 0x96, 0x3e, 0x0f | EFI\_TCP4\_PROTOCOL.GetModeData() **–** invokes GetModeData()with none of the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with none of the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.33 | 0x05f9a5f1, 0x445d, 0x46d2, 0xb8, 0x82, 0xf0, 0xe2, 0x34, 0x72, 0xca, 0x48 | EFI\_TCP4\_PROTOCOL.GetModeData() **–**invokes GetModeData()to correctly get the Tcp4ConfigData.TypeOfService. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to open an active connection,then receive the packet.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with none of the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.34 | 0x529c2a7a, 0xf533, 0x4777, 0xa3, 0x7d, 0x09, 0x6f, 0x0c, 0x52, 0x99, 0xa7 | EFI\_TCP4\_PROTOCOL.GetModeData() **–**invokes GetModeData()to correctly get the Tcp4ConfigData and TimeToLive. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to open an active connection,then receive the packet.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with none of the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.35 | 0xe6bc773d, 0xf461, 0x4f0f, 0x97, 0xed, 0x78, 0x69, 0x7f, 0x0b, 0x81, 0xcb | EFI\_TCP4\_PROTOCOL.GetModeData() **–**invokes GetModeData()to correctly get the Tcp4ConfigData and AccessPoint. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to open an active connection,then receive the packet.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with none of the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.1.36 | 0x42f51ebd, 0x24d2, 0x42af, 0xb9, 0xad, 0x7e, 0xb2, 0xfe, 0x2a, 0x18, 0x65 | EFI\_TCP4\_PROTOCOL.GetModeData() **–**invokes GetModeData()to correctly get the Tcp4ConfigData and AccessPoint. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to open an active connection,then receive the packet.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() with none of the optional parameters Tcp4State, Tcp4ConfigData, Ip4ModeData, *MnpConfigData* and *SnpModeData* value of NULL. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.2.1 | 0x64729d75, 0x1007, 0x4b20, 0x9b, 0x78, 0x59, 0xc4, 0xc7, 0x02, 0xec, 0x9e | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure()when using a default address, and configuration has not finished yet. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure()when using a default address, and configuration (through  DHCP, BOOTP, RARP, etc.) has not finished yet. The return status should be **EFI\_NO\_MAPPING**.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.2 | 0xe8cef00f, 0x0796, 0x4b1c, 0xbd, 0x09, 0x2c, 0x86, 0xdb, 0x4d, 0xba, 0x44 | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure()with a *TcpConfigData->AccessPoint.*StationAddress value of an invalid unicast IPv4 address when *TcpConfigData->AccessPoint.UseDefaultAddress*is **FALSE**. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call **Tcp.**Configure()with a *TcpConfigData->AccessPoint.*StationAddressvalue of an invalid unicast IPv4 address when  *TcpConfigData->AccessPoint.UseDefaultAddress*is **FALSE**. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.3 | 0x6aaabbca, 0xb7d3, 0x49a1, 0x8f, 0x11, 0x4a, 0x82, 0x3f, 0x2e, 0xd9, 0x00 | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure() with a *TcpConfigData->AccessPoint.*SubnetMask value of an invalid IPv4 address mask when *TcpConfigData->AccessPoint.UseDefaultAddress* is **FALSE**. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call **Tcp.**Configure()with a *TcpConfigData->AccessPoint.*SubnetMaskvalue of an invalid IPv4 address mask when*TcpConfigData->AccessPoint.UseDefaultAddress*is **FALSE**. The subnet mask must be contiguous. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.4 | 0xa176de8a, 0xd68d, 0x4529, 0x97, 0xb5, 0xcf, 0x13, 0xa7, 0xe3, 0x33, 0xc0 | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure() with a *TcpConfigData->AccessPoint. RemoteAddress* value of an invalid unicast IPv4 address. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure()with a *TcpConfigData->AccessPoint. RemoteAddress*value of an invalid unicast IPv4 address. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.5 | 0xf3f1b054, 0xd497 ,0x4e1a, 0xa4, 0x67, 0x9c, 0x23, 0xab, 0xbb, 0x43, 0x08 | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure() when a same access point has been configured in other TCP instance previously. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() when a same access point has been configured in other TCP instance previously. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.6 | 0x6fd9c85c, 0x7cc5, 0x480f, 0xa9, 0x14, 0x8f, 0xbd, 0x0d, 0x30, 0xba, 0x15 | EFI\_TCP4\_PROTOCOL. Configure() **–** invokes Configure() with a *TcpConfigData->AccessPoint.RemoteAddress* value of 0 when *TcpConfigData->AccessPoint.ActiveFlag*is **TRUE**. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call **Tcp.**Configure() with a *TcpConfigData->AccessPoint.RemoteAddress* value of 0 when *TcpConfigData->AccessPoint.ActiveFlag*is **TRUE**.  The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.7 | 0x0782f91f, 0x5553, 0x4854, 0x92, 0xbe, 0xb5, 0x25, 0x79, 0x0b, 0x42, 0x79 | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure() with a *TcpConfigData->AccessPoint.RemotePort* value of 0 when *TcpConfigData->AccessPoint.ActiveFlag*is **TRUE**.. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() with *TcpConfigData->AccessPoint.RemotePort* is 0 when *TcpConfigData->AccessPoint.ActiveFlag*is **TRUE**.  The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.8 | 0x21e9706f, 0xf449, 0x4c3c, 0x95, 0x6e, 0xf4, 0x28, 0xdd, 0x22, 0x5a, 0xb9 | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure() with the TCP instance configured without calling Configure() with NULL to reset it. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure()to configure the new Tcp4 instance.  3. Call EFI\_TCP4\_PROTOCOL.Configure()to configure the Tcp4 instance again without calling Configure() with NULL to reset it. The return status should be **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.2.9 | 0xa1e6077c, 0x035e, 0x4684, 0x81, 0xe2, 0x99, 0xb2, 0x44, 0x4e, 0x0b, 0x9d | EFI\_TCP4\_PROTOCOL.Configure() **–** invokes Configure()when one or more of the control options are not supported in the implementation. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure()when one or more of the control options are not supported in the implementation. The return status should be **EFI\_UNSUPPORTED**.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### Connect()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.4.1 | 0x0dc45007, 0xff6e, 0x41da, 0x81, 0x05, 0x55, 0x2d, 0x88, 0xe8, 0x09, 0x14 | EFI\_TCP4\_PROTOCOL.Connect() **–** invokes Connect()when the instance has not been configured. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Connect()when the instance has not been configured. The return status should be **EFI\_NOT\_STARTED**.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.4.2 | 0xa00efef2, 0xd596, 0x4332, 0xa1, 0x9b, 0x38, 0x0a, 0xe0, 0xd7, 0x23, 0xe0 | EFI\_TCP4\_PROTOCOL.Connect() **–** invokes Connect()when the instance is not configured as an active one. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as not an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to open a connection when the instance is not configured as an active one. The return status should be **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.4.3 | 0xe204e699, 0x7941, 0x4d65, 0x8b, 0x2e, 0xf2, 0xbe, 0xd3, 0x6c, 0xcf, 0x7e | EFI\_TCP4\_PROTOCOL.Connect() – invokes Connect() when the instance is not in Tcp4StateClosed state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()configure the instance againwhen it is not in Tcp4State**Closed** state. The return status should be **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.4.4 | 0x3011f8f5, 0x6ccf, 0x46f4, 0xb9, 0x9a, 0x09, 0xd0, 0xf3, 0xde, 0x3a, 0x12 | EFI\_TCP4\_PROTOCOL.Connect() **–** invokes Connect()with a ConnectionToken value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()with a ConnectionToken value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.4.5 | 0x513b33c4, 0x4df0, 0x449e, 0xb8, 0xf5, 0xd6, 0x4e, 0x30, 0x27, 0x0e, 0xa4 | EFI\_TCP4\_PROTOCOL.Connect() **–** invokes Connect() with a ConnectionToken->CompletionToken.Event value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect()with a ConnectionToken->CompletionToken.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.4.6 | 0x672d833 2, 0xa9a0, 0x4111, 0xa2,0x95, 0x10,0xfe, 0x88,0x17, 0x86,0x04 | EFI\_TCP4\_PROTOCOL.Connect() – Connect()  must return EFI\_CONNECTION\_REFUSED when the instance is  in *SYN-RCVD* state & receive a *RST* | 1. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new  Tcp4 child.  2. Call  EFI\_TCP4\_PROTOCOL.Configure()  to configure the new instance.  3. Call  EFI\_TCP4\_PROTOCOL.Connect()  Receive SYN & Send a SYN to put TCP state machine in SYN-RCVD state.  4. Send a RST & check Connection Token state to be changed to **EFI\_CONNECTION\_REFUSED**  4. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()to destroy the created Tcp4 child and clean up the environment. |

### Accept()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.5.1 | 0x81d93128, 0xfcda, 0x49fa, 0x87, 0xea, 0xd4, 0x8e, 0x83, 0x1a, 0x6e, 0x8b | EFI\_TCP4\_PROTOCOL.Accept() **–** invokes Accept()when the instance has not been configured. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Accept()when the instance has not been configured. The return status should be **EFI\_NOT\_STARTED**.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.2 | 0x9f46e8f3, 0xc4e0, 0x4027, 0x88, 0x09, 0x6b, 0xc4, 0xc6, 0x5d, 0xca, 0xf5 | EFI\_TCP4\_PROTOCOL.Accept() **–** invokes Accept()when the instance is not a passive one. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()when the instance is not a passive one. The return status should be  **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.3 | 0xd59b4f29, 0x874c, 0x4282, 0xac, 0x7d, 0x3f, 0xf6, 0x8d, 0x52, 0x54, 0xe8 | EFI\_TCP4\_PROTOCOL.Accept() – invokes Accept() when the instance is not in Tcp4StateListen state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept() to initiate an asynchronous accept request to wait for an incoming connection.  4.Call EFI\_TCP4\_PROTOCOL.GetModeData() to change the instance state to Tcp4StateEstablished.  5. Call EFI\_TCP4\_PROTOCOL.Accept()when the instance is not in Tcp4State**Listen** state. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.4 | 0x85f6ab8a, 0x9374, 0x4afe, 0x85, 0x76, 0x5e, 0xa4, 0x44, 0x57, 0x87, 0x31 | EFI\_TCP4\_PROTOCOL.Accept() **–** invokes Accept()when the same listen token has already existed in the listen token queue of this TCP instance. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept() to initiate an asynchronous accept request to wait for an incoming connection.  4. Call EFI\_TCP4\_PROTOCOL.Accept()again when the same listen token has already existed in the listening token queue of this TCP instance. The return status should be **EFI\_ACCESS\_DENIED**.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.5 | 0x26f62b3c, 0xb67a, 0x4f2a, 0x86, 0x8f, 0x65, 0x30, 0xf6, 0x5e, 0xe3, 0x1b | EFI\_TCP4\_PROTOCOL.Accept() – invokes Accept() with a ListenToken value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept()with a ListenTokenListenToken value of NULL. The return status should be EFI\_INVALID\_PARAMETEREFI\_INVALID\_PARAMETER.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.6 | 0x4fbd5006, 0x0d81, 0x40d0, 0xb8, 0xff, 0xca, 0x77, 0x03, 0x80, 0x34, 0xb6 | EFI\_TCP4\_PROTOCOL.Accept() **–** invokes Accept()with a ListentToken->CompletionToken.Event value ofNULL**.** | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept()with a ListentToken->CompletionToken.Event value ofNULL**.** The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.7 | 0x0df289ca, 0xfc53, 0x4fc2, 0x92, 0xb3, 0xb4, 0x3a, 0xcf, 0x3c, 0x50, 0x34 | EFI\_TCP4\_PROTOCOL.Accept() **–** invokes Accept()to listen on the passive instance to accept an incoming connection request. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept()to listen on the passive instance to accept an incoming connection request. The return status should be EFI\_SUCCESS.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.5.8 | 0x71f6d2e2, 0x9d2a, 0x435e, 0x83,0x0e, 0x63,0x9f, 0x1f,0xe7, 0x31,0x95 | EFI\_TCP4\_PROTOCOL.Accept() –Call Accept()to listen on the passive instance to  accept an incoming connection request. If received a RST, parent TCP State should Still be LISTEN. | 1. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild()to create a new Tcp4 child.  2. Call  EFI\_TCP4\_PROTOCOL.Configure()to  configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept()  to listen on the passive instance to accept  an incoming connection request.  4.Send a RST to Host and Call GetModeData() to get Parent state. The state should be LISTEN  5. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()to destroy the  created Tcp4 child and clean up the  environment. |
| 5.25.1.5.9 | 0x0b1d8b5c, 0xc111, 0x4548, 0xac,0x9e, 0x3c,0xc2, 0x85,0xaa, 0x0d,0xab | EFI\_TCP4\_PROTOCOL.Accept()–Call Accept()  to listen on the passive instance to  accept an incoming connection request. Must return EFI\_SUCCESS after a successful passive mode connection | 1. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild()to create a new  Tcp4 child.  2. Call  EFI\_TCP4\_PROTOCOL.Configure()to  configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept()to listen on the passive instance to accept  an incoming connection request.  4.Connect & check return status should be EFI\_SUCCESS.  5. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the  created Tcp4 child and clean up the  environment. |
| 5.25.1.5.10 | 0xbef6d443,0xbece, 0x4315, 0x84,0x57, 0x90,0xe4, 0xb1,0xc4, 0x34,0x0a | EFI\_TCP4\_PROTOCOL.Accept()–Call Accept()to listen on the passive instance to  accept an incoming connection request. New created connection state should be ESTABLISED after a successful passive mode connection | 1. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild()to create a new  Tcp4 child.  2. Call  EFI\_TCP4\_PROTOCOL.Configure() to  configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept()to listen on the passive instance to accept  an incoming connection request.  4.Connect & Call GetModeData() check new created connection status should be ESTABLISED.  5. Call  EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()to destroy the  created Tcp4 child and clean up the environment. |

### Transmit()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.6.1 | 0xe268c41a, 0x3749, 0x4e6c, 0x95, 0xdc, 0x11, 0x6c, 0x4a, 0x57, 0x93, 0x40 | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() when the instance has not been configured. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Transmit() when the instance has not been configured. The return status should be **EFI\_NOT\_STARTED**.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.2 | 0xf05cb723, 0x7194, 0x45f9, 0xae, 0x3d, 0x52, 0x9b, 0xb3, 0x63, 0xde, 0x19 | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() to transmit a packet with a Token value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with a Token value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.3 | 0xaaba9e1f, 0xdc0c, 0x4320, 0x8a, 0x01, 0x51, 0xc0, 0x07, 0x22, 0xfb, 0x73 | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() to transmit a packet with a Token->CompletionToken.Event value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with a Token->CompletionToken.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.4 | 0x96eb6c53, 0x68bc, 0x4a3b, 0xa4, 0x07, 0x96, 0xbc, 0x97, 0xac, 0x8e, 0x1e | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit()Transmit() to transmit a packet with a Token->Packet.TxDatavalue of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with a Token->Packet.TxDatavalue of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.5 | 0xc0bce6b7, 0xcd60, 0x484a, 0xb3, 0x37, 0xf5, 0xb4, 0xfe, 0x99, 0x30, 0xb2 | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() to transmit a packet with a Token*->Packet.FragmentCount*value of 0**.** | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with a Token*->Packet.FragmentCount*value of 0**.** The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.6 | 0xc00b7871, 0xa4ac, 0x4bfd, 0x81, 0xda, 0x78, 0x52, 0xc0, 0xc0, 0x54, 0x65 | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() to transmit a packet with a Token*->Packet.DataLength* value other than equal to the sum of fragment lengths. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with a Token*->Packet.DataLength* value other than equal to the sum of fragment lengths. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.7 | 0x7e824bb2, 0xb6cd, 0x49b6, 0x9f, 0x1b, 0xe3, 0x60, 0x02, 0x7d, 0xd7, 0x5f | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() when a transmit completion token with the same Token->CompletionToken.Event which was already in the transmission queue. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() when a transmit completion token with the same Token->CompletionToken.Event in step 4 which was already in the transmission queue. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.8 | 0x97d1f634, 0x39aa, 0x44a3, 0xb4, 0xc8, 0x22, 0xa4, 0x17, 0x2b, 0x9a, 0x12 | EFI\_TCP4\_PROTOCOL.Transmit() – invokes Transmit() when the current instance is in Tcp4StateClosed state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. OS send RST to let EUT enter Tcp4StateClosed state.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() when the current instance is in Tcp4State**Closed** state. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.9 | 0x42145b1a, 0xdd0c, 0x40f8, 0x8f, 0x9a, 0x4c, 0xfc, 0xb6, 0xde, 0x88, 0x2e | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() when the current instance is a passive one and it is in Tcp4State**Listen** state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Transmit() when the current instance is a passive one and it is in Tcp4State**Listen** state. The return status should be **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.10 | 0xb1618c99, 0xc9c4, 0x4b90, 0x86, 0x4a, 0x8f, 0xa3, 0x32, 0xfd, 0x13, 0xe6 | EFI\_TCP4\_PROTOCOL.Transmit() **–** invokes Transmit() when user has called Close() to disconnect this connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() the disconnect the connection opened in step 3.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet when the connection was disconnected in step 4. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.11 | 0xb5b0f9ab, 0x04f3, 0x4269, 0x96, 0xa6, 0x40, 0xf5, 0x48, 0xa0, 0x9b, 0x7e | EFI\_TCP4\_PROTOCOL.Transmit() **–** Tests that the [EUT] correctly handles FIN segment during data transmission. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet with FIN, ACK segment to end one side of the connection.  6. call ReceiveTcpPacket to receive the packet, and send the ack packet.  7. call ReceiveTcpPacket to receive the packet, and send the ack packet for the second time.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.12 | 0x19052fce, 0x5744, 0x470f, 0x8f, 0xc0, 0xc3, 0x84, 0xcc, 0x88, 0x57, 0x1d | EFI\_TCP4\_PROTOCOL.Transmit() **–**Checks the validity of [PSH] bit during data transimission, by sending 16 bytes data segment to [EUT], with [ENTS] default MSS = 536. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet.  5. Check the Token*.Status*.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.13 | 0x7740ac88, 0x4cf3, 0x4943, 0x9b, 0xf9, 0xec, 0xc4, 0x6a, 0x58, 0xcc, 0x90 | EFI\_TCP4\_PROTOCOL.Transmit() **–**Checks the validity of [PSH] bit during data transimission, by sending 1024 bytes data segment to [EUT], with [ENTS] default MSS = 536. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet.  5. Check the Token*.Status.*  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.14 | 0xc6e11d01, 0x485b, 0x4585, 0x9a, 0x2e, 0xcf, 0x43, 0xac, 0x94, 0x2e, 0x1a | EFI\_TCP4\_PROTOCOL.Transmit() **–**Transmits two fragments. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet.  5. Check the Token*.Status*.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.6.15 | 0xa5f63716, 0xd4a2, 0x44dc, 0x93, 0x2a, 0xd8, 0xdf, 0x33, 0xd2, 0xa1, 0x65 | EFI\_TCP4\_PROTOCOL.Transmit() **–**Transmits more fragments. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet.  5. Check the Token*.Status*.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### Receive()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.7.1 | 0xe28b3623, 0xc8ba, 0x431a, 0x91, 0xcd, 0xe2, 0xc5, 0x60, 0x36, 0xaa, 0x80 | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() when the instance has not been configured. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Receive() when the instance has not been configured.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.2 | 0x484c93a6, 0x93ba, 0x429f, 0x9e, 0x63, 0x0a, 0x7d, 0x5c, 0x19, 0xf5, 0xc7 | EFI\_TCP4\_PROTOCOL.Receive – invokes Receive() with a Token value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() with a Token value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.3 | 0xbe0ff6c1, 0x26a0, 0x4c3f, 0x88, 0xc7, 0xcc, 0xfc, 0x9f, 0xc8, 0xbe, 0x28 | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() with a Token->CompletionToken.Event value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() with a Token->CompletionToken.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.4 | 0xd0d81b11, 0x23dc, 0x41ac, 0x8c, 0xec, 0xdd, 0x3c, 0x0f, 0x9f, 0x25, 0xef | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() with a Token*->Packet.RxData*value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() with a Token*->Packet.RxData*value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.5 | 0x6d723765, 0x1345, 0x45ad, 0xb3, 0x57, 0xf0, 0xbc, 0xa1, 0x4c, 0x0c, 0x8f | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() with a Token->Packet.RxData->DataLengthvalue of 0. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() with a Token->Packet.RxData->DataLengthvalue of 0. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.6 | 0x1aed8f61, 0xf658, 0x4abb, 0xac, 0x90, 0x04, 0x74, 0x2c, 0x46, 0x87, 0x57 | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() with a Token*->Packet.RxData->DataLength*is not the sum of all FragmentBuffer length in FragmentTable. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() with a Token*->Packet.RxData->DataLength*value other than the sum of all FragmentBuffer length in FragmentTable. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.7 | 0x2ac8bc18, 0x6c65, 0x4b0d, 0xaf, 0xf1, 0x4f, 0xb5, 0x2e, 0x63, 0xc8, 0x4f | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() when the receive completion token with the same Token->CompletionToken.Event was already in the receive queue. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.  5. Call EFI\_TCP4\_PROTOCOL.Receive() again when the receive completion token with the same Token->CompletionToken.Event was already in the receive queue. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.8 | 0x77f0240a, 0x16a4, 0x471a, 0x95, 0x52, 0xf6, 0x58, 0xf9, 0xbb, 0x11, 0xb1 | EFI\_TCP4\_PROTOCOL.Receive – invokes Receive() when the current instance is in Tcp4StateClosed state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. OS send RST segment to let EUT enter Tcp4State**Closed** state.  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet when the instance is in Tcp4State**Closed** state**.** Thereturn status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.9 | 0x276a8e6d, 0xf79a, 0x4cc5, 0xba, 0xcb, 0x99, 0x48, 0x38, 0x59, 0xde, 0xfb | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() when the current instance is a passive one and it is in  Tcp4StateListen state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Accept() to accept a connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet when the instance is a passive one and it is in Tcp4State**Listen** state. Thereturn status should be **EFI\_ACCESS\_DENIED**.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.10 | 0xdde96586, 0xd067, 0x4f04, 0xa0, 0xd9, 0xbd, 0x94, 0x0e, 0x30, 0x97, 0x90 | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() when user has called Close() to disconnect this connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() the disconnect the connection opened in step 3.  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet when the connection was disconnected in step 4. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.20 | 0xc527d95b, 0xbf72, 0x4c94, 0xa8, 0xcc, 0x60, 0x8c, 0x47, 0x04, 0x85, 0x07 | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() when the communication peer has closed the connection and there is no any buffered data in the receive buffer of this instance. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Handles the three-way handshake.  5. Configure the OS side to initiate the connection closing.  6. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.  7. Clean up the environment on EUT side. |
| 5.25.1.7.21 | 0xc9109f21, 0xd490, 0x4382, 0xbb, 0x22, 0x12, 0xfd, 0x81, 0x67, 0x14, 0xec | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() fails when connection is reseted by the communication peer**.** | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Handles the three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.  6. Configure the OS side to reset the connection.  7. Clean up the environment on EUT side. |
| 5.25.1.7.11 | 0x36f08e10, 0xbf24, 0x4a97, 0x83, 0x42, 0x99, 0x32, 0x33, 0xff, 0xbe, 0x18 | EFI\_TCP4\_PROTOCOL.Receive – invokes Receive() to receive a packet. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet and then check the Token*.Status* to verify if the data has been transmitted successfully. The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.12 | 0xda1653b3, 0xcf85, 0x4152, 0x88, 0x30, 0xd4, 0xbf, 0x54, 0x17, 0x6a, 0x22 | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() to receive a packet with two fragment data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet with two fragment data,and then check the Token*.Status* to verify if the data has been transmitted successfully. The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.13 | 0xd40ff5f0, 0xcb1d, 0x41cf, 0x8e, 0xab, 0x3f, 0xce, 0xa8, 0x93, 0x3f, 0x4f | EFI\_TCP4\_PROTOCOL.Receive **–** invokes Receive() to receive a packet with ten fragment data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet with ten fragment data,and then check the Token*.Status* to verify if the data has been transmitted successfully. The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.14 | 0xf1974d5d, 0x5860, 0x4519, 0x8b, 0x8f, 0x78, 0xce, 0x0a, 0xad, 0xbb, 0xec | EFI\_TCP4\_PROTOCOL.Receive **–** Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(no overlaps). | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.15 | 0xc9d79086, 0x5eb8, 0x4c76, 0xa4, 0xc4, 0xf1, 0xfe, 0x78, 0x6f, 0xc0, 0x31 | EFI\_TCP4\_PROTOCOL.Receive **–**Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the second head overlaps the first tail). | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.16 | 0x3c0cc77e, 0xfb9b, 0x4b24, 0x85, 0xd0, 0xaf, 0x3f, 0x39, 0xc8, 0xfd, 0xb7 | EFI\_TCP4\_PROTOCOL.Receive **–**Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the second segment is included in the middle of the first one). | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.17 | 0x5252cae8, 0xb23b, 0x456e, 0x97, 0xdf, 0x1c, 0x01, 0xdd, 0xc4, 0xcd, 0x05 | EFI\_TCP4\_PROTOCOL.Receive **–**Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the third segment is included in the head of the second one). | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.18 | 0x8a11bbca, 0xe267, 0x4221, 0xa5, 0x50, 0x33, 0x62, 0x33, 0x88, 0xeb, 0x06 | EFI\_TCP4\_PROTOCOL.Receive **–**Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the third segment is included in the middle of the second one). | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.7.19 | 0x794eff7b, 0xb88f, 0x4f67, 0x9d, 0xa1, 0xd5, 0x0e, 0xa6, 0xbc, 0x5c, 0x37 | EFI\_TCP4\_PROTOCOL.Receive **–**Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the first and the second segment is joined by the third one). | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet.The return status should be EFI\_SUCCESS.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### Close()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.8.1 | 0xc92fad2d, 0x446d, 0x43d7, 0xaf, 0xbe, 0x81, 0xce, 0x03, 0xd4, 0xe8, 0x12 | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() when the instance has not been configured. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Close() to close a connection when the instance has not been configured. The return status should be **EFI\_NOT\_STARTED**.  3. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.8.2 | 0x82827716, 0xb622, 0x4527, 0xb8, 0x9e, 0xa5, 0x30, 0x59, 0xce, 0xc9, 0xec | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() when Configure() has been called with *TcpConfigData* set to NULLand this function has not returned. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() with *TcpConfigData* set to NULL.  3. Call EFI\_TCP4\_PROTOCOL.Close() when the Configure() function has not returned. The return status should be **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.8.3 | 0x9f19e873, 0x71a5, 0x4350, 0xa6, 0xb5, 0xa9, 0x96, 0x8c, 0x64, 0xe6, 0xde | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() when the previous Close() call on this instance has not finished. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() to disconnect the connection opened in step 3.  5. Call EFI\_TCP4\_PROTOCOL.Close() when the previous Close() call on this instance has not finished. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.8.4 | 0xa9472aa1, 0xfff1, 0x4130, 0x90, 0xc9, 0xf8, 0x87, 0x69, 0x8f, 0x8b, 0xc1 | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() with a CloseToken value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() with a CloseToken value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.8.5 | 0x09caa34e, 0xdf4f, 0x4dcf, 0xbe, 0x5b, 0x7b, 0xe3, 0xf3, 0x68, 0x90, 0xc0 | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() with a CloseToken->CompletionToken.Event value of NULL. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() with aCloseToken->CompletionToken.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.8.6 | 0x3756329a, 0x21c3, 0x41c6, 0xa1, 0x03, 0x15, 0x9a, 0x57, 0x93, 0x8e, 0x9f | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() as function test. After user called Configure() with NULL without close stopping, the *Close*Token*.Completion*Token*.Status* should be **EFI\_ABORTED**. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() to disconnect the connection opened in step 3. The return status should be EFI\_SUCCESS.  5. Call EFI\_TCP4\_PROTOCOL.Configure() with NULL without close stopping, then verify the *Close*Token*.Completion*Token*.Status*tobe **EFI\_ABORTED**.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.8.7 | 0x499852f9, 0x49c2, 0x4168, 0x8f, 0x90, 0xab, 0x97, 0x0f, 0x06, 0x53, 0x0b | EFI\_TCP4\_PROTOCOL.Close **–** invokes Close() as function test. Abort the TCP connection on close instead of the standard TCP close process by setting the *AbortOnClose* to **TRUE**. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection.  4. Call EFI\_TCP4\_PROTOCOL.Close() to disconnect the connection opened in step 3 with *AbortOnClose*set to **TRUE**. The return status should be EFI\_SUCCESS. Then verify Token*.Status* has been updated to **EFI\_ABORTED**.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### CnntClosing

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.13.1 | 0xc9fa5b59, 0x7a1c, 0x4b2b, 0x9b, 0xce, 0x6b, 0xad, 0x38, 0x12, 0x2b, 0x0d | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the closing connection when it initiates the closing. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow, then check the Token*.Status* to verify the connection has been closed.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.2 | 0x8ae1e58b, 0xcd65, 0x4fb0, 0xba, 0x12, 0x43, 0x95, 0xef, 0xab, 0x9c, 0xd1 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the closing connection when [OS] initiates the closing. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Configure the [OS] to initiate the connection closing.  5. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow, then check the Token*.Status* to verify the connection has been closed.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.3 | 0x8b1bcbd7, 0x3db6, 0x46ec, 0x8b, 0xf0, 0x84, 0xb4, 0xb9, 0x0f, 0xb8, 0x95 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the simultaneous closing connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow, then check the Token*.Status* to verify the connection has been closed.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.4 | 0xebc0e165, 0x3146, 0x4fa1, 0x9a, 0xd8, 0x6d, 0x56, 0xdf, 0xb0, 0x9f, 0xd6 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the reception of normal data segments after having already received partner's FIN segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Configure the [OS] to initiate the connection closing. Then configure the [OS] to send data segments to the [EUT].  5. Call Tcp.GetModeData(), and there is a expectation that EUT should return to CLOSE state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.5 | 0x9530e11a, 0x4d42, 0x4c45, 0x9e, 0xe9, 0x30, 0x82, 0xfc, 0xc9, 0x0f, 0x97 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly handle the reception of unacceptable data segments after having already received partner's FIN segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Configure the [OS] to initiate the connection closing. Then configure the [OS] to send data segments to the [EUT].  5. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow, then check the Token*.Status* to verify the connection has been closed.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.6 | 0x8cb38a66, 0xfb72, 0x4dce, 0x94, 0x8b, 0x3e, 0x8f, 0xae, 0x66, 0x6f, 0x98 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly perform the retransmission of FIN segment during the connection closing process. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow. EUT should timeout 3 times and follow the sequence: ,6,12 ...then check the Token*.Status* to verify the connection has been closed.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.7 | 0xc9ef7a67, 0xc2a7, 0x40b4, 0xa9, 0x31, 0xba, 0x7a, 0x83, 0x16, 0x53, 0x15 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the half-close of the communication peer. If your peer still wants to send data after sending out **FIN**, EUT should ignore the data and interact with the peer correctly. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. OS get the transmitted data packet and respond with **FIN**, **ACK** segment to end one side of the connection.  6. Expand the receive window together with data in the segment, EUT should ignore the data and interact with the peer correctly.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.8 | 0xc4e81c62, 0xe709, 0x4096, 0xbb, 0xfb, 0x59, 0x99, 0x07, 0xaf, 0x89, 0x82 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly support partner's half-close. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Configure the [OS] to initiate the connection closing. Then Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. OS get the transmitted data packet and check the Token*.Status* to verify the data has been sent out.  6. Call EFI\_TCP4\_PROTOCOL.Close() to close the connection, then check the Token*.Status* to verify the connection has been closed.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.9 | 0x37b8e036, 0x3ff9, 0x4401, 0x81, 0x76, 0xa5, 0x70, 0xd9, 0x16, 0xa9, 0x4e | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly wait a **2xMSL** timeout period while it has initiated the closing of a connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow.  5. Call EFI\_TCP4\_PROTOCOL.Connect() to reopen the connection when [EUT] is still in **TIME-WAIT** state. The return status should be **EFI\_ACCESS\_DENIED**.  6. Check the Token*.Status* to verify the connection has been closed.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.10 | 0x2c9f0ffe, 0xf355, 0x4a2f, 0xb6, 0xa2, 0xbf, 0x84, 0x6c, 0xe8, 0x33, 0x2f | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle a valid SYN segment while it is in **TIME-WAIT** state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow.  5. Send a **SYN** segment with a larger sequence number than the previous connection contained. If the **SYN** is not in the window, an **ACK** should be sent out.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.11 | 0xaaf0c2ad, 0x5433, 0x46cf, 0xa4, 0xd9, 0xc3, 0xea, 0x65, 0xe1, 0x38, 0xfc | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the buffered receive data when application already performed active close. The buffered data should be removed and **RST** segment should be sent out. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake. Configure the [OS] to send data segments to the [EUT].  4. Call EFI\_TCP4\_PROTOCOL.Close() to close the connection. The [EUT] should send out a RST segment.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.12 | 0x7996049d, 0xc63f, 0x4bb4, 0x96, 0xa2, 0xb1, 0x90, 0xe7, 0x35, 0x8c, 0x3c | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the send buffered data when application has already performed active close. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake.  4. Create event and configuration for transmit and close interface invoking.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Then [OS] get the transmitted data packet.  6. Call EFI\_TCP4\_PROTOCOL.Close() to close the connection. Then configure the [OS] to interact data transmission with the [EUT].  The last segment should have the **FIN** flag set.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.13 | 0xa740c41c, 0xa9b1, 0x4194, 0x8a, 0xf5, 0x6c, 0x92, 0xd9, 0x20, 0xc7, 0x78 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle and receive the data segment in **<SYN>** and **<FIN, ACK>** segments, receive all the data (throw down a receive token) after data transmission finished. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Configure OS to send data together with FIN flag set. Then Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data sent with the **SYN** and **<FIN, ACK>** segment.  Check the received segment data length.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.13.14 | 0xd012d6bb, 0x9dac, 0x4e3b, 0xa5, 0x54, 0xf6, 0xe9, 0xf5, 0x77, 0x22, 0xb4 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle and receive the data segment in **<SYN>** and **<FIN, ACK>** segments, and receive all the data (throw down a receive token) before data transmission. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data sent with the **SYN** and **<FIN, ACK>** segment. Then configure OS to send data together with FIN flag set. Check the received segment data length.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### CnntOpening

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.14.1 | 0x156e08bb, 0x21c4, 0x48a0, 0xbe, 0xc0, 0x8d, 0x0c, 0x17, 0x7b, 0x90, 0xf2 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly receive and handle the SYN segment with data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data sent with the **SYN** segment. Then check the received segment data length.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.2 | 0xd7814ee7, 0x2cc3, 0x4cc6, 0xb4, 0x3c , 0x54, 0x7e, 0x1f, 0x73, 0xc3, 0x12 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through active open. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the passive TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.3 | 0xeac7fe49, 0x5202, 0x457f, 0x9e, 0x77, 0x49, 0xe5, 0x77, 0xa1, 0x4b, 0x4e | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through active open. This connection should not affect any previously established connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the active TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child for the second connection..  5. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the second instance as an active one  6. Call EFI\_TCP4\_PROTOCOL.Connect() for the second active TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.4 | 0xc5678e42, 0x6d91, 0x41c1, 0x96, 0x2d, 0xb6, 0x7b, 0xaa, 0x72, 0xf8, 0x21 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through passive open with unspecified address/port pair. This connection should not affect any previously established connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify if the connection has been established.  4. Try to establish TCP connection with unspecified address/port pair.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.5 | 0x3131d110, 0x7545, 0x46c5, 0x91, 0xd1, 0x87, 0x01, 0xd3, 0x04, 0x7f, 0xcf | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through passive open with specified address/port pair. This connection should not affect any previously established connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive TCP instance, then handles the three-way handshake. Check the Token*.Status* to verify if the connection has been established.  4. Try to establish TCP connection with unspecified address/port pair.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.6 | 0x165ad06c, 0xf630, 0x4516, 0x95, 0xba, 0x90, 0x3f, 0xd8, 0xa2, 0x4d, 0xe4 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. It performs the following interactions:  A ------<SYN>------> B  A <-----<SYN>------- B  A --<SYN, ACK>--> B  A <-----<ACK>------- B | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the active TCP instance, then handle the three-way handshake. Check the Token*.Status* to verify if the connection has been established.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.7 | 0x2328abeb, 0x2dca, 0x4960, 0xa0, 0x93, 0x42, 0x94, 0xc8, 0x8c, 0x3d, 0x51 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. This connection should not affect any previously established connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, then handle the three-way handshake and check the Token*.Status* to verify the connection has been established.  4. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child for the second connection.  5. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the second instance as an active one.  6. Call EFI\_TCP4\_PROTOCOL.Connect() for the second instance, then handle the three-way handshake and check the Token*.Status* to verify if the connection has been established.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.9 | 0xe39e864a, 0x347d, 0x4c08, 0xa7, 0xec, 0x0e, 0x55, 0x34, 0xe8, 0xa0, 0x20 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly time out when waiting a TCP connection to be established in SYN\_SENT state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, and during 60 seconds, EUT should timeout following the sequence: 3, 6, 12, 24….  4. Check the Token*.Status* to verify the connection has been timeouted.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.10 | 0x697d126d, 0xd496, 0x448b, 0x85, 0x08, 0x60, 0x6d, 0xc1, 0xc6, 0x3f, 0x65 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly time out when waiting a TCP connection to be established in SYN\_SENT state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, and during 60 seconds, EUT should timeout following the sequence: 3, 6, 12, 24….In addition, EUT should send out RST segment and return to CLOSED state.  4. Check the Token*.Status* to verify the connection has been timeouted.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.12 | 0xb22365c7, 0x6daa, 0x48e9, 0xa3, 0x7a, 0x1d, 0xe5, 0x47, 0xf4, 0x04, 0x4e | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. It performs the following interactions:  A ------<SYN>------> B  A <-----<SYN>------- B  A ----<SYN, ACK>---> B  A <---<SYN, ACK>---- B | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, then handle the three-way handshake.  4. Check the Token*.Status* to verify the connection has been established.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.13 | 0x8e4d9bac, 0x42b6, 0x408f, 0xa2, 0x44, 0xd3, 0xfe, 0x9b, 0xdc, 0x0c, 0xc7 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. It performs the following interactions:  A ------<SYN>------> B  A <-----<SYN>------- B  A <---<SYN, ACK>---- B  A ----<SYN, ACK>---> B | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, then handle the three-way handshake.  4. Check the Token*.Status* to verify the connection has been established.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.14 | 0x72d8a37d, 0x312e, 0x44ee, 0x86, 0xcb, 0xb5, 0x58, 0x5c, 0x63, 0x6d, 0x65 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly receive and handle the <SYN, ACK> segment with data, throw down receive token after data transmission. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, then handle the three-way handshake.  4. Check the Token*.Status* to verify the connection has been established.  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data sent with the SYN segment. Then check the segment data length.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.15 | 0xe0c87d8a, 0x81d4, 0x4634, 0xa2, 0x0a, 0xee, 0xba, 0xdc, 0x44, 0x96, 0xe6 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly receive and handle the <SYN, ACK> segment with data, throw down receive token before data transmission. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data sent with the SYN segment.  4. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, then handle the three-way handshake.  5. Check the Token*.Status* to verify the connection has been established.  6. Get the received segment datalength to check the correction.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.16 | 0x13f5c5e1, 0xd4dc, 0x437d, 0xac, 0xa2, 0x93, 0x1a, 0x8d, 0x85, 0xe0, 0xd3 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the flag combination: ACK, FIN through active open. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the new instance, then handle the three-way handshake. In addition, EUT should ignore this unexpected segment and retransmit the SYN segment.  4. Check the Token*.Status* to verify the connection has been established.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.17 | 0x656575ec, 0x018b, 0x475a, 0x80, 0xa0, 0xff, 0x32, 0xef, 0x50, 0x31, 0x74 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the flag combination: FIN, ACK through passive open. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the new instance, then handle the three-way handshake. In addition, the data sent together with the FIN,ACK segment should be processed.  4. Check the Token*.Status* to verify the connection has been established.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.18 | 0xcaba9876, 0xc926, 0x42b3, 0xaf, 0x99, 0xb5, 0x7d, 0x71, 0x83, 0x62, 0x20 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly handle the flag combination: SYN, FIN, ACK through passive open. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the new instance, then handle the three-way handshake.  4. Handle the normal three-way handshake. Then check the Token*.Status* to verify the connection has been established.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.14.19 | 0xcd97a722, 0xc8fe, 0x4584, 0xb3, 0x9c, 0x65, 0x9b, 0xbb, 0x2c, 0x5a, 0x6f | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that can correctly refuse the attempted connections from broadcast and multicast address. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the new instance, then handle the three-way handshake.  4. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### CongestionCtrl

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.15.1 | 0xb0cdf9b2, 0x0cc0, 0x4e99, 0x96, 0x83, 0xde, 0xf3, 0x96, 0xc1, 0xc6, 0xa7 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly perform the slow start at the beginning of the connection transmission. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. OS get the transmitted data packet and interact with EUT to expand the cwnd.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.15.2 | 0x05d19fac, 0x66e6, 0x4f41, 0xba, 0x70, 0xff, 0x3e, 0x48, 0x7f, 0x4d, 0x4a | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly perform the slow start and congestion avoidance algorithms when data segment timeout causes congestion. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. OS gets the transmitted data segments of the fist stage, and check the token status of transmit interface, then begin the second stage data transmission.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments as the second stage.  7. Wait for data retransmission and send back the ACK to all the transmitted data segments. In addition, EUT should enter slow start.  8. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.15.4 | 0xc12b24da, 0xa3c5, 0x4820, 0x81, 0x98, 0x6e, 0x34, 0xad, 0x28, 0xfc, 0xaf | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly perform the slow start and congestion avoidance algorithms when **SYN** segment timeout causes congestion. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept() to for the instance. 4. Handle the three-way handshake. Configure the [OS] to ignore the first **SYN** segment and wait for the *ConnectionTimeout* seconds. When received the second **SYN** segment, make the [OS] send back the **SYN**, **ACK** segment.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  6. OS get the transmitted data packet and interact with EUT to expand the cwnd. In addition, check the token status of transmit interface.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.15.5 | 0xf5c35856, 0x3c84, 0x40ce, 0xba, 0xf4, 0x91, 0x57, 0x7e, 0xfa, 0x44, 0x98 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly performs the fast retransmit and fast recovery algorithms receiving 3 or above duplicated acknowledgements. When an ACK arrives that acknowledges new data, this ACK is Full acknowledgements. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. OS get the transmitted data segments of the first stage, and check the token status of transmit interface, then begin the second stage data transmission.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet as the second stage.  6. The cwnd should be expanded to 11\*SMSS after the 1st stage data transmission. The second stage of data transmission includes 8192 (16\*MSS) bytes data. Configure the OS to generate consecutive duplicate ACKs.  7. Configure the OS to acknowledge the last data segment and EUT will end the fast recovery and enter the congestion avoidance again.  8. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.15.6 | 0x0df29ac1, 0x5b58, 0x49cc, 0x95, 0x31, 0xce, 0xce, 0xb4, 0x49, 0xb5, 0x3a | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly generate duplicated acknowledgements when it received disordering segments. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Configure the OS to send consecutive data segments to the EUT, drop one segment in the middle and EUT should generate duplicated ACKs as the result of receiving every data segments.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.15.7 | 0x3a4fb624, 0x8b05, 0x46ce, 0x97, 0xd7, 0x0f, 0xc9, 0x1e, 0x5d, 0x37, 0x6a | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly performs the fast retransmit and fast recovery algorithms receiving 3 or above duplicated acknowledgements. After exiting the fast recovery, [EUT] should enter congestion avoidance. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments of the first stage.  5. OS get the transmitted data segments of the fist stage, check the token status of transmit interface. Then begin the second stage data transmission.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments of the second stage.  7. The cwnd should be expanded to 11\*SMSS after the 1st stage data transmission. The second stage of data transmission includes 8192 (16\*MSS) bytes data. Configure the OS to generate consecutive duplicate ACKs.  8. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments of the third stage. The third stage of data transmission should perform congestion avoidance.  9. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.15.8 | 0xa4d6bd97, 0x6d30, 0x4fec, 0x8b, 0x50, 0xcf, 0xac, 0xb1, 0x7e, 0x9e, 0x0a | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] that correctly performs the NewReno modification to TCP's fast recovery algorithm. After the first fast recovery, when an ACK arrives that acknowledges new data, this ACK is partial acknowledgements. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments of the first stage.  5. OS get the transmitted data segments of the fist stage, check the token status of transmit interface. Then begin the second stage data transmission.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments of the second stage.  7. The cwnd should be expanded to 11\*SMSS after the 1st stage data transmission. The second stage of data transmission includes 8192 (16\*MSS) bytes data. Configure the OS to generate consecutive duplicate ACKs.  8. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### NagleSWSA

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.16.1 | 0xceef47a7, 0xf194, 0x4200, 0x9a, 0xbc, 0xe2, 0x9d, 0xfe, 0x80, 0xaa, 0x49 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly disables the Nagle Algorithm. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a small segment.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit another small segment.  6. OS gets the first transmitted data packet, and the 2nd segment should be sent out immediately.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.2 | 0x3906f7fa, 0xbe7b, 0x435a, 0xb6, 0x78, 0x1d, 0x5b, 0xba, 0xe5, 0x51, 0x4a | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly disables the Nagle Algorithm. When retransmission happens, the accumulated small segments should be sent out together. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a small segment.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit another small segment.  6. As Nagle is disabled, the two segments should be sent out immediately. In addition, they should be sent out separately during retransmission.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.3 | 0xa528b7a1, 0x23cb, 0x4601, 0xb2, 0x74, 0xd7, 0x0b, 0xcc, 0x17, 0x5e, 0x42 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the small segments in accord with Nagle algorithm. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and enable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit three small segment.  5. OS get the first transmitted data segment and send back ACK segment. As Nagle is enabled, the last two segments should be sent out together.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.4 | 0x0d5581c0, 0x6903, 0x4387, 0xaf, 0xf7, 0xe3, 0x2c, 0xac, 0x17, 0xee, 0x33 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the small segments in accord with Nagle algorithm. When retransmission happens, the accumulated small segments should be sent out together. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and enable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit three small segment.  5. OS get the first transmitted data segment and as Nagle is enabled, the last two segments should be sent out together.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.5 | 0xabf756ac, 0x54a7, 0x492c, 0xae, 0xa6, 0x6d, 0x46, 0xd7, 0x44, 0xb8, 0x72 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. In a stream of full-sized segments there should be an ACK for at least every second segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. Configure the [OS] to send 10 full-sized data segments. There should be at least an ACK for every second segment.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.6 | 0x94c3ee05, 0x142e, 0x4f2e, 0x8a, 0x9a, 0x8f, 0x05, 0x25, 0xbb, 0xb4, 0x83 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. A TCP should implement a delayed ACK, but an ACK should not be excessively delayed. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. EUT should delay ACK the data segment, but the delay MUST be less than 0.5 second.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.7 | 0x81d74381, 0xb0df, 0x4ef3, 0x8a, 0x1c, 0xdc, 0x7b, 0xe9, 0x60, 0xc6, 0x02 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. In a stream of single-byte segments there should be an ACK for at least every second segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. Configure the [OS] to send 20 single-byte data segments. There should be at least an ACK for every second segment.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.8 | 0xd7c7813e, 0x4624, 0x4f11, 0xb3, 0x65, 0x45, 0x6e, 0x00, 0x30, 0x30, 0xe2 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The receiver should not advertise a larger window until the window can be increased at least one full-sized segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as a passive one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Accept() for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. Configure the [OS] to send 4 data segment to fill the receive buffer.  5. Call Receive interface to get one full-sized data.  6. Get the Window expansion segment. Then send another 1024-bytes data to refill the EUT receive buffer.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.9 | 0xf853dee2, 0xa900, 0x417b, 0xb5, 0xce, 0x80, 0x86, 0x55, 0x17, 0xab, 0x57 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP sender. The sender should not transmit unless everything can be sent out and no need to wait ACK. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and enable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the active instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a small segment.  5. OS gets the EUT transmitted data segment. In addition, EUT should send out all the left data segments.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.10 | 0x93015811, 0x2c00, 0x4834, 0x83, 0x17, 0x7b, 0xbf, 0x7f, 0x1a, 0xcb, 0x52 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The sender should not transmit unless everything can be sent out and Nagle algorithm is disabled. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the active instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a small segment.  5. OS gets the EUT transmitted data segment. In addition, configure the OS to acknowledge the second segment and advertise enough window to let EUT transmit all the left data segments.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.11 | 0xfa149507, 0x1607, 0x44da, 0xb2, 0xae, 0x5f, 0xd3, 0x51, 0x7d, 0x82, 0xba | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The sender should not transmit unless a full-sized segment can be sent. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the active instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a small segment.  5. EUT should set persist timer, configure OS to increase the window size to exceed 512 bytes before the persist timer times out. In addition, repeat the steps before finishing the data transmission.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.16.12 | 0xceb5c9e5, 0xebce, 0x4486, 0xb5, 0xc5, 0x06, 0xa6, 0x0c, 0x36, 0x5e, 0xa6 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The sender should not transmit unless at least one-half of the Max Window that receive ever advertised. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one, and disable the Nagle control option.  3. Call EFI\_TCP4\_PROTOCOL.Connect() for the active instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a small segment.  5. EUT should set persist timer, configure OS to increase the window size by 256 octets consecutively. Make sure the windows size exceed one-half of the Max Window that receive ever advertised before persist timer times out.  6. Increase the windows size step by step, when it accesses the left data size, EUT should send out the left buffered data at one time.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### UrgHandling

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.12.1 | 0x355d3648, 0x8375, 0x4b16, 0x94, 0xc4, 0x19, 0xe1, 0xbc, 0x87, 0xfc, 0x8b | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly uses the urgent pointer to denote the last urgent octet of urgent data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments.  5. Get the transmitted data segment and check the urgent pointer, it should point to the sequence number of the last octet. Then check the token status of transmit interface.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.2 | 0x03663fa9, 0x0a34, 0x43a5, 0x84, 0x5b, 0x2c, 0x36, 0x7f, 0x7e, 0xb6, 0xd8 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly uses the urgent pointer to denote the last urgent octet of urgent data. The urgent data exceeds the maximum number of urgent pointer. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments with the length 65536.  5. Get the transmitted data segment and check the urgent pointer.  6. The urgent pointer will rollback but the EUT should maintain the correct value of the urgent pointer. After sending out the first data segment, EUT should send the second data segment with urgent pointer 65024(65536 – 512).  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.3 | 0xfce0e13a, 0x35df, 0x4713, 0xaf, 0xb8, 0x4d, 0x1e, 0xcc, 0xa5, 0x82, 0x9b | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly uses the urgent pointer to denote the last urgent octet of urgent data. The urgent pointer rollbacks for two times. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit data segments with the length 131401.  5. OS get the transmitted data packet and interact with EUT to expand the cwnd.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.4 | 0x75f47641, 0x2982, 0x4d51, 0x95, 0x3b, 0x4b, 0x65, 0x91, 0x73, 0x5e, 0x76 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. OS sends some urgent data between normal data transmission. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Configure the OS to send normal data including urgent data segments.  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the normal data and get the received segment data length to check the correction.  6. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the first section of urgent data. Get the received segment data length to check the correction.  7. Send the remained urgent data and normal data.  8. Call EFI\_TCP4\_PROTOCOL.Receive()to receive the second section of urgent data and the remained normal data. Check the data length.  9. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.5 | 0xd0f54967, 0xaa9b, 0x4017, 0x87, 0x87, 0x24, 0xfb, 0x34, 0x9d, 0xe4, 0x51 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. OS sends some urgent data in the SYN segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the normal data and get the received segment data length to check the correction.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.6 | 0x4cbb57e5, 0xe348, 0x4340, 0x81, 0x9e, 0xed, 0x61, 0x5a, 0xc2, 0x1a, 0x35 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. The urgent pointer just points to the sequence of FIN flag. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Configure the OS to send normal data including urgent data segments.  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data segments, and check the data length.  6. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the urgent data segments, and check the data length.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.7 | 0x6145a7f3, 0xbb3d, 0x48e8, 0xab, 0xdf, 0x90, 0xc9, 0x87, 0x82, 0xdc, 0x25 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. The urgent pointer exceeds the sequence of FIN flag. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Configure the OS to send normal data including urgent data segments, and make the urgent pointer exceed the sequence if FIN flag..  5. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the data segments, and check the data length.  6. Call EFI\_TCP4\_PROTOCOL.Receive() to receive the urgent data segments, and check the data length.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.8 | 0x73cf4c9a, 0x8c1d, 0x4b7f, 0x94, 0x7c, 0x7f, 0x74, 0x06, 0xf5, 0x10, 0x1d | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the urgent data transmission when communication peer's receive window is 0. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. After OS got the transmitted data packet, Make the [OS] send an acknowledge segment with a 0 window. Then check whether EUT can still send out data segment or not.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.12.9 | 0x6019f57b, 0xd99f, 0x47b4, 0x94, 0x4a, 0x86, 0x80, 0x3e, 0x55, 0x63, 0x54 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the urgent data transmission when communication peer's receive window is 0. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. After OS got the transmitted data packet, Make the [OS] send an acknowledge segment with a 0 window.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit an urgent packet. Then check whether EUT can still send out data segment or not.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### RstHandling

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.17.1 | 0x1dd96986, 0x44c7, 0x4981, 0xba, 0x01, 0x14, 0x73, 0xff, 0x82, 0xb2, 0xed | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <CLOSED> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow.  5. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate RST generation in <CLOSED> state.  6. In <CLOSED> state, check OS send SYN, and EUT respond with RST.  7. In <CLOSED> state, check OS send FIN, and EUT respond with RST.  8. In <CLOSED> state, check OS send URG|ACK, EUT respond with RST.  9. In <CLOSED> state, check OS send RST|ACK, and EUT respond with Nothing.  10. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.2 | 0x554f2d12, 0xfa71, 0x48eb, 0x96, 0x02, 0xff, 0x5c, 0xfb, 0x8d, 0x45, 0xe6 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <ESTABLISHED> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate RST generation in <ESTABLISHED> state.  5. Instruct OS send out un-acceptable ACK, and expect receive ACK which indicate the expected next sequence number.  6. Verify <EUT> send out ACK, and the recvd ACK.ack\_id indicating correct seq\_id.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.3 | 0x12dea7e9, 0x1773, 0x4adb, 0x97, 0x27, 0xe8, 0xc3, 0xcf, 0xfb, 0xb9, 0x7b | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <CLOSE-WAIT> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Change the state from ESTABLISEHD to CLOSE\_WAIT, and call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it.  5. Verify <EUT> send out ACK, and the recvd ACK.ack\_id indicating correct seq\_id. Then send RST to disconnect the session  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.4 | 0xebf00938, 0xb335, 0x4a33, 0xa2, 0x7b, 0x4d, 0x54, 0xf6, 0x42, 0x72, 0x99 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <LAST-ACK> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Instruct EUT enter LAST\_ACK state:  OS --> EUT: FIN  EUT --> OS: ACK  EUT --> OS: FIN  Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow.  5. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it has enter LAST\_WAIT state.  6. Verify whether EUT correctly send out RST in LAST\_ACK state.  7. Verify does connection remains in the same states after received any unacceptable segment.  8. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.5 | 0x21941c4e, 0xb4e3, 0x422b, 0x81, 0x58, 0xef, 0xcd, 0x28, 0xb0, 0xee, 0xef | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <FIN\_WAIT\_1> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Instruct EUT from ESTABLISHED to LAST\_ACK state: Call EFI\_TCP4\_PROTOCOL.Close() interface to do a graceful close working flow. Then call EFI\_TCP4\_PROTOCOL.GetModeData() to validate enter FIN\_WAIT\_1 state.  5. Verify whether EUT correctly send out RST in FIN\_WAIT\_1 state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.6 | 0xee1c295d, 0x13e1, 0x4bc3, 0x94, 0x4b, 0xb5, 0x2e, 0xaf, 0x48, 0xb2, 0x5f | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <FIN\_WAIT\_2> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Instruct EUT from ESTABLISHED to LAST\_ACK state: Call EFI\_TCP4\_PROTOCOL.Close() interface to do a graceful close working flow, then OS --> EUT: ACK. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate enter FIN\_WAIT\_1 state.  5. Verify whether EUT correctly send out RST in FIN\_WAIT\_1 state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.7 | 0x4fac9b90, 0xf3c4, 0x4779, 0xab, 0x3f, 0x32, 0xe8, 0xd9, 0x9b, 0x8b, 0x09 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <CLOSING> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Instruct EUT enter LAST\_ACK state:  (1) EUT --> OS: FIN  Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow.  (2) OS --> EUT: FIN  (3) EUT --> OS: ACK  (4) Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in CLOSING state.  5. Verify whether EUT correctly send out RST in CLOSING state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.8 | 0xfa9a7729, 0xc10b, 0x4233, 0xb8, 0xe9, 0xeb, 0x8a, 0xf6, 0x65, 0x85, 0x75 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the reset segment while in <TIME\_WAIT> state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Instruct EUT enter LAST\_ACK state:  (1) EUT --> OS: FIN  Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow.  (2) EUT --> OS: FIN  (3) OS --> EUT: FIN|ACK  (4) Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in TIME\_WAIT state.  5. Verify whether EUT correctly send out RST in TIME\_WAIT state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.9 | 0xd6646a78, 0x5508, 0x4643, 0x9d, 0x9b, 0x0c, 0xca, 0x22, 0x22, 0x0a, 0xc6 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly send out the empty Acknowledge segment after received data segment with unacceptable Acknowledge. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configured the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in <ESTABLISHED> state.  5. Validate RST generation in <ESTABLISHED> state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.10 | 0xc0b6a498, 0x1cbd, 0x4df0, 0x97, 0x71, 0xd1, 0x95, 0x14, 0xec, 0x74, 0xf2 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly handles the reception of a RST segment in LISTEN state - <EUT> should ignore the reset segment and remain in LISTEN state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in LISTEN state.  4. Instruct <OS> send a RST segment, and expect behavior: no response from EUT.  5. Instruct <OS> send a SYN segment, and receive SYN|ACK from Ack.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.11 | 0xe48e5518, 0xaf29, 0x4e2b, 0xb9, 0xba, 0xfe, 0xfc, 0x0a, 0x37, 0x19, 0x56 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly handles the reception of a RST segment in SYN\_RCVD state - Previous state is LISTEN and it returns to LISTEN state | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Instruct <OS> send a SYN segment, and expect behavior: receive SYN|ACK. Then receive the packet.  4. Instruct <OS> send a valid RST segment,  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in LISTEN state.  5. Re-initialize the connection, and let it enter SYN\_RCVD state.  6. Instruct <OS> send a SYN segment, and expect behavior: receive SYN|ACK. Then receive the packet.  7. Instruct <OS> send a valid RST segment,  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in LISTEN state.  8. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in LISTEN state.  9. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.13 | 0x386fc38f, 0x8f4d, 0x4c34, 0x85, 0x68, 0x62, 0x71, 0x51, 0x0c, 0x35, 0xf5 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly handles the reception of a RST segment in SYN\_SENT state - return to CLOSED state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to initialize connection.  4. <EUT> --> <OS>: SYN, then call EFI\_TCP4\_PROTOCOL.**GetModeCall()** to validate it is in SYN\_SENT state.  5. Instruct <OS> send a valid RST segment, and its sequence number is one-byte less than window boundary. Expect that on receiving a valid RST, the connection returned to CLOSED state.  6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.14 | 0xb886e8c2, 0xf6e7, 0x40e3, 0xbf, 0xc8, 0x78, 0xc3, 0x91, 0x91, 0x8d, 0xae | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly handles the reception of a RST segment in ESTABLISHED state - return to CLOSED state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to initialize connection.  4. <EUT> --> <OS>: SYN  <OS> --> <EUT>: SYN|ACK  <EUT> --> <OS>: ACK  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in ESTABLISHED state.  5. Instruct <OS> send a valid RST segment, and its sequence number is one-byte less than window boundary. Expect that on receiving a valid RST, the connection returned to CLOSED state.  6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.15 | 0x1a49bc31, 0xad75, 0x4165, 0xaf, 0xff, 0xae, 0xf0, 0x1d, 0x1a, 0x7b, 0x29 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly handles the reception of a RST segment in FIN\_WAIT\_1 state - return to CLOSED state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to initialize connection.  4. <EUT> --> <OS>: SYN  <OS> --> <EUT>: SYN|ACK  <EUT> --> <OS>: ACK  Call EFI\_TCP4\_PROTOCOL.Close() to make EUT enter FIN\_WAIT\_1.  <EUT> --> <OS>: FIN  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in FIN\_WAIT\_1 state.  5. Instruct <OS> send a valid RST segment, and its sequence number is at window boundary. Expect that on receiving a valid RST, the connection returned to CLOSED state.  6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.16 | 0xe88fa39a, 0xfbc5, 0x4366, 0x9c, 0x68, 0x48, 0x99, 0x78, 0xd4, 0x0e, 0x23 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly handles the reception of a RST segment in FIN\_WAIT\_2 state - return to CLOSED state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to initialize connection.  4. <EUT> --> <OS>: SYN  <OS> --> <EUT>: SYN|ACK  <EUT> --> <OS>: ACK  Call EFI\_TCP4\_PROTOCOL.Close() to make EUT enter FIN\_WAIT\_1.  <EUT> --> <OS>: FIN  <OS> --> <EUT>: ACK  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in FIN\_WAIT\_2 state.  5. Instruct <OS> send a valid RST segment, and its sequence number is what is expected. Expect that on receiving a valid RST, the connection returned to CLOSED state.  6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.17 | 0x600a697d, 0x6250, 0x49a2, 0x97, 0xac, 0xa3, 0xc7, 0x28, 0x20, 0x3f, 0x9d | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in SYN\_SENT state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect() to initialize connection.  4. <EUT> --> <OS>: SYN, then call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in SYN\_SENT state.  5. Instruct <OS> send a invalid RST segment, and RST.ack doesn't ack the SYN. Then call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is still in SYNC\_SENT state.  6. OS --> EUT: SYNC  EUT --> OS: SYNC\_ACK  EUT --> OS: RST, and validate the RST.seq be equal to received ACK.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.18 | 0xa9631841, 0x2e5e, 0x49cb, 0xb9, 0xeb, 0x9a, 0xba, 0x04, 0xaf, 0xa3, 0x5f | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in LISTEN state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.GetModeData()to validate it is in LISTEN state.  4. Instruct <OS> send a invalid RST segment, RST.Seq not in the window. In addition, expect that no packet send out from EUT.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.19 | 0x4226ee2f, 0xd8f2, 0x46e2, 0x8f, 0xaf, 0x1a, 0x00, 0x42, 0xf6, 0x7e, 0x29 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in LISTEN state. |  |
| 5.25.1.17.20 | 0xdf8dc924, 0xa0a4, 0x4520, 0x9d, 0x07, 0x59, 0xae, 0x21, 0x8b, 0xb4, 0x53 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in ESTABLISHED state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in ESTABLISHED state.  4. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection will still in ESTABLISHED state.  5. OS --> EUT: SYNC, and expect:  EUT --> OS: SYNC\_ACK  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.21 | 0x17f9536e, 0xa472, 0x4b33, 0x9e, 0x2c, 0x30, 0xb1, 0x8d, 0x82, 0x49, 0x44 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in FIN\_WAIT\_1 state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to make EUT enter FIN\_WAIT\_1. Then call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it.  5. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in FIN\_WAIT\_1 state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.22 | 0xe99b76fc, 0x1f57, 0x4f68, 0x8b, 0x16, 0x4e, 0xf8, 0x1a, 0xa7, 0xc6, 0x01 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in FIN\_WAIT\_2 state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake.  4. **Call** EFI\_TCP4\_PROTOCOL.Close() to make EUT enter FIN\_WAIT\_1. Then OS --> EUT: ACK, and call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it enter in FIN\_WAIT\_2 state.  5. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in FIN\_WAIT\_2 state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.23 | 0xc7f281cf, 0x5ff7, 0x475e, 0xab, 0x0e, 0x8e, 0x13, 0x76, 0xb4, 0x46, 0xa6 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in CLOSE\_WAIT state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake.  4. <OS> --> <EUT>: FIN  <EUT> --> <OS>: ACK  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate it is in CLOSE\_WAIT state.  5. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in CLOSE\_WAIT state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.24 | 0xeea6dd88, 0x1df4, 0x438e, 0xa5, 0x2b, 0xee, 0x9f, 0xc5, 0xb2, 0xd6, 0xf7 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in CLOSEING state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to close the connection; <EUT> --> <OS>: FIN; <OS> --> <EUT>: FIN. Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate enter CLOSEING state.  5. Instruct <OS> send an invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in CLOSEING state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.25 | 0xb316e0cc, 0x260e, 0x4d24, 0xa5, 0xee, 0xf4, 0xae, 0x34, 0x30, 0xa9, 0x52 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in TIME\_WAIT state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to close the connection;  <EUT> --> <OS>: FIN;  <OS> --> <EUT>: FIN.  <EUT> --> <OS>: ACK;  <OS> --> <EUT>: ACK;  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate enter TIME\_WAIT state.  5. Instruct <OS> send an invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in TIME\_WAIT state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.17.26 | 0x9a8293c3, 0x3d43, 0x4cfd, 0xb3, 0x73, 0xb1, 0xca, 0x0d, 0xef, 0x91, 0x66 | EFI\_TCP4\_PROTOCOL **–** Tests that the <EUT> correctly validate the rcvd RST segment while in LAST\_LACK state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()to initialize connection. Then Handle three-way handshake.  4. <OS> --> <EUT>: FIN;  <EUT> --> <OS>: ACK;  Call EFI\_TCP4\_PROTOCOL.Close() to close the connection;  <EUT> --> <OS>: FIN  Call EFI\_TCP4\_PROTOCOL.GetModeData() to validate enter LAST\_LACK state.  5. Instruct <OS> send an invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in LAST\_LACK state.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### WinFlowCtrl

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.18.1 | 0xe107339e, 0xed3b, 0x44fa, 0xa9, 0x18, 0x83, 0xf0, 0x10, 0x0e, 0x70, 0x14 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives the segment that has the advertised receive window open right-edge and close left-edge. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  5. After OS got the transmitted data packet, configure the [OS] to send back ACK segment to acknowledge the first segment and keep the advertised window to be 1536 octets.  6. Configure the [OS] to finish the data interaction with [EUT].  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.2 | 0x823c66d7, 0x2787, 0x400d, 0x8f, 0x62, 0x69, 0xdd, 0x3b, 0x21, 0x1f, 0x58 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives the segment that has the advertised receive window open right-edge and keep left-edge. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 3072.  5. After OS got the transmitted data packet, configure the [OS] to send back ACK segment to acknowledge the first segment and change the advertised window to be 1024 octets.  6. Acknowledge the SYN segment sent from the [EUT] and change the advertised window to be 1536 octets.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.3 | 0x530d5e6d, 0x928e, 0x42c3, 0xa4, 0x6e, 0x74, 0x93, 0xc0, 0xac, 0xca, 0xbf | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives the segment that has the advertised receive window open right-edge and include the duplicated ACKs. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 5120.  5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the SYN segment sent by the [EUT].  6. Change the advertised window to be 2048 octets and capture the responded segments.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.4 | 0x1a697687, 0x3deb, 0x4b7b, 0x89, 0x6f, 0x78, 0x35, 0x95, 0x1b, 0x7a, 0xe9 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly transmits the advertised window size of data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 1024.  5. After OS got the transmitted data packet, configure the [OS] to send back ACK segment to acknowledge the first segment and keep the advertised window to be 2048 octets.  6. Configure the [OS] to finish the data interaction with [EUT].  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.5 | 0xbc12abb0, 0xf022, 0x4705, 0x9d, 0x12, 0x32, 0x78, 0xaa, 0x89, 0x80, 0xb8 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Make the [OS] send ten full-sized and consecutive segments and capture the responded segments. The [EUT] should not respond with an acknowledgement segment for each of the received segments. There should be an acknowledgement segment for at least every second segment.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.6 | 0x0541c800, 0x7639, 0x46f5, 0x90, 0x1a, 0x20, 0x7c, 0xc3, 0x11, 0x44, 0xc9 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles a link partner's shrinking window with right-edge shrinking and left-edge closing - test Right Edge Shrinks with Left Edge Closes. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 1024.  5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the data segments and change the advertised window to be 1024 octets and capture the responded segments.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.7 | 0x613c599e, 0x26e8, 0x4d39, 0x96, 0xe1, 0x2d, 0x30, 0xd6, 0xbe, 0x20, 0xf7 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles a link partner's shrinking window with right-edge shrinking and left-edge closing - test Right Edge Shrinks with Left Edge Keeps. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 5120.  5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the SYN sent by the [EUT] and change the advertised window to be 2048 octets and capture the responded segments.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.8 | 0xbbb555fc, 0x8a4d, 0x41eb, 0xaf, 0x1d, 0x8c, 0xc9, 0x87, 0xb4, 0x46, 0x45 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles a link partner's shrinking window with right-edge shrinking and left-edge closing - test Right Edge Shrinks with Duplicated ACK. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 5120.  5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the SYN sent by the [EUT] and change the advertised window to be 2048 octets and capture the responded segments. In addition, window update segment including duplicated ACKs should be discarded  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.9 | 0x5b42c4d0, 0xaf0c, 0x4ae9, 0x9f, 0xfc, 0xb4, 0xf8, 0x3f, 0xcd, 0x4d, 0x73 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles a link partner's shrinking window when the data retransmission happens. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 5120.  5. When capturing the retransmitted A segment, configure the [OS] to send back ACK segments and capture the responded segments separately. The ACK is to acknowledge the A segment and change the advertised window to be 1536 octets.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.10 | 0xf3a8f990, 0x0f1f, 0x408f, 0xad, 0x66, 0x2c, 0x98, 0x1f, 0xc1, 0x65, 0x34 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives data segments while its partner's advertised window is 0. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 5120.  5. After OS got the transmitted data packet, make the [OS] send an acknowledge segment with a 0 window. Then validate EUT send out the ACK segment correctly.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.11 | 0xce6f5d62, 0x0c72, 0x412d, 0x9a, 0xf4, 0xc8, 0xcc, 0x96, 0x8d, 0xe3, 0x42 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly probes a partner's advertised 0 window. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet with the length 5120.  5. After OS got the transmitted data packet, make the [OS] send an acknowledge segment with a 0 window, and in current implementation, the 0 window probing segment contains no data. Then validate EUT send out the ACK segment correctly.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.18.12 | 0x0165a4f8, 0x5976, 0x4051, 0xa2, 0x73, 0x9e, 0xa1, 0x62, 0xe5, 0xc9, 0xac | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly probes a partner's advertised 0 window, when partner advertises non-0 window, EUT can send out left data segments correctly. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. Configure the [OS] to send tcp segment with different length payloads. Then validate EUT process and respond correctly.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### Options

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.19.1 | 0x1f1c574b, 0xd5b8, 0x4111, 0x90, 0x14, 0xf6, 0x50, 0x04, 0x3c, 0x8a, 0x71 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly ignores the unsupported options as long as the option has a valid length field. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. Configure the [OS] to send different unsupported options' tcp segments.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.2 | 0x5be584cc, 0x39e0, 0x4bcf, 0xaf, 0x69, 0xda, 0x64, 0xff, 0xfa, 0x9a, 0x02 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles End-of-Options option. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  4. Configure the [OS] to send tcp segment with CombinedOptions containing End-of-Options option.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.3 | 0xbfc4a76f, 0x19ad, 0x4f34, 0x97, 0x51, 0x07, 0xd3, 0xd5, 0xe4, 0x92, 0x0a | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles End-of-Options option. There are more options behind the End-of-Options option. These options should be ignored. | 1. Build combined options field as No-Option No-Option No-Option End-of-Options Option MSS10-Option (this option should be ignored).  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.4 | 0xbc3c725e, 0x8784, 0x4559, 0x81, 0x91, 0x60, 0x66, 0x93, 0xb0, 0x9a, 0xd1 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles No-Operation option, segment with the No-Operation option between multiple options but not coinciding with the word boundary. | 1. Build combined options field as the No-Operation option between multiple options but not coinciding with the word boundary.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Then check OS get the transmitted data packet.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.5 | 0x957bd7ef, 0x6a40, 0x46e2, 0xbd, 0x62, 0x9b, 0xa2, 0x39, 0x35, 0xe2, 0x96 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles No-Operation option, segment with the No-Operation option between multiple options at the word boundary. | 1. Build combined options field as the No-Operation option between multiple options at the word boundary.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Then check OS get the transmitted data packet.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.6 | 0xd4f6ab22, 0x5d0a, 0x4f9e, 0xa5, 0xce, 0x80, 0x94, 0xf2, 0x42, 0xc2, 0xd0 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles No-Operation option, segment with the No-Operation option between multiple options but not coinciding with the word boundary. one item of the same option is split in different words. | 1. Build combined options field as the No-Operation option between multiple options at the word boundary, one item of the same option is split in different words.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Then check OS get the transmitted data packet.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.7 | 0xee9c7ea4, 0x3bec, 0x4de0, 0x84, 0x65, 0xcb, 0x18, 0x21, 0x4e, 0x3b, 0x01 | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality - Tests that the [EUT] correctly transmits MSS option in <SYN> segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance.  4. Handle three-way handshake and check EUT send out SYN segment with MSS correctly.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.8 | 0xd69abe03, 0xdbb5, 0x473f, 0x91, 0x59, 0xf3, 0x43, 0xe7, 0xf0, 0x04, 0xe8 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly receives MSS option in <SYN> segment, and then replies to transmit MSS option in <SYN, ACK> segment correctly. | 1. Build TCP segment with MSS OPTION, here MSS = 256.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  5. Handle three-way handshake and Check the Token*.Status* to verify the Accept connection has been completed.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.9 | 0x98e61624, 0x7c30, 0x4d11, 0x8b, 0xf0, 0x45, 0x4e, 0xd8, 0x0b, 0x21, 0xc0 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly adheres to the MSS of the connection. [EUT] will automatically divide up transmitting data segment if its size is larger than [OS] announced MSS value. | 1. Build TCP segment with MSS OPTION, here MSS = 100.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.**Connect ()** for the new instance. Then handle three-way handshake.  5. [OS] send SYN & ACK segment with MSS option and receive ACK segment.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.10 | 0x50efbcf2, 0xabe6, 0x4cfa, 0x94, 0xc7, 0x78, 0x86, 0xe9, 0x38, 0xd8, 0x59 | EFI\_TCP4\_PROTOCOL **–** Tests that when [EUT] received <SYN> segment without MSS option, [EUT] could take [OS]'s MSS as RFC default value 536. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handle three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  5. [OS] sends data to [EUT]: Create a data segment to be transmitted, with size larger than RFC\_TCP\_DEF\_MSS.  6. Call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.11 | 0xab7715ef, 0x8d1f, 0x4b68, 0xb8, 0x66, 0xb3, 0x8d, 0x84, 0x71, 0x98, 0x71 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly transmit and receive the MSS option in segments without the SYN flag set high. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  4. Handle three-way handshake. Send segment with another MSS in non-SYN segment. The [EUT] should ignore the MSS option in non-SYN segments.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.12 | 0xa0845af3, 0x382f, 0x4ab9, 0x8d, 0xe0, 0xe6, 0xc3, 0x0c, 0xcd, 0x95, 0xd0 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the reception of MSS option with invalid option value. Let MSS = 0. Value 0 should be ignored and replaced with 64 (EFI\_TCP\_MIN\_MSS). | 1. Build TCP MSS option, MSS = 0, invalid value.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance.Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.13 | 0xa7d40772, 0xc53a, 0x44f6, 0x98, 0x1e, 0xbf, 0x9f, 0xa6, 0xcf, 0x56, 0x5b | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the reception of MSS option with invalid option value. Let MSS > 1460. [EUT] should ignore MSS larger than 1460 and replace it with 1460. | 1. Build TCP MSS option, MSS = 2048, invalid value, larger than 1460(Maximum MSS).  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance.Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.14 | 0x1b50447f, 0x868c, 0x4ea4, 0x93, 0xc0, 0xcb, 0x00, 0x73, 0x31, 0x52, 0xcf | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the reception of segments with unaligned MSS option. Format 1. | 1. Create unaligned MSS option with format 1.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Handle three-way handshake.  5. [OS] send SYN & ACK segment with MSS option. Then call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token**.Status** to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.15 | 0x3973bbb2, 0xe1c5, 0x40ea, 0x8e, 0x50, 0xdb, 0x53, 0x8e, 0xc1, 0x42, 0xa9 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the reception of segments with unaligned MSS option. Format 2. | 1. Create unaligned MSS option with format 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Handle three-way handshake.  5. [OS] send SYN & ACK segment with MSS option. Then call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token**.Status** to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.16 | 0xb74382c6, 0x37dc, 0x4151, 0x9d, 0xe3, 0xd4, 0x98, 0x8e, 0x4c, 0xd8, 0xcd | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handle the reception of segments with unaligned MSS option. Format 3. | 1. Create unaligned MSS option with format 3.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Handle three-way handshake.  5. [OS] send SYN & ACK segment with MSS option. Then call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token**.Status** to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.17 | 0x53cd1a49, 0xaa07, 0x4bf8, 0x95, 0x45, 0xa4, 0xd3, 0x83, 0x6c, 0x4f, 0xb4 | EFI\_TCP4\_PROTOCOL **–** Tests that when [EUT] received <SYN> segment without MSS option, [EUT] could take [OS]'s MSS as RFC default value 536. With unaligned window scale option as format 2. | 1. Create TCP option. Windows Scale: shift.cnt = 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.18 | 0x454d5884, 0xf7e1, 0x43a8, 0x97, 0xab, 0x48, 0xbb, 0xd2, 0x22, 0xa6, 0x5b | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly turns window scale option on. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.19 | 0xc8d0492a, 0x79e8, 0x411c, 0x91, 0x42, 0x08, 0x2e, 0x7a, 0x81, 0xbb, 0x86 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly ignores a Window scale option in a segment without SYN bit set. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Then create another TCP option with another Windows Scale Value which will be sent in <ACK> Segment.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.20 | 0x691e1119, 0xe737, 0x4560, 0x96, 0x33, 0xb7, 0x57, 0xd6, 0x2e, 0x22, 0xde | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly interacts with the partner that doesn’t support window scaling option. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handle three-way handshake.  4. OS send DATA & ACK segment, then call EFI\_TCP4\_PROTOCOL.Receive() to receive a packet. In addition, check the Token*.Status* to verify the data has been transmitted successfully.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.21 | 0xca16dc5d, 0x5720, 0x45d0, 0xa2, 0xe4, 0x19, 0x98, 0xc2, 0xa8, 0x5f, 0x5c | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the segment with window scaling shift count exceeding 14. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Calculate [OS] MAX acceptable window. . In addition, set window scale with 16.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.22 | 0xade14e0f, 0xa957, 0x4489, 0x83, 0xf8, 0xdb, 0x9f, 0x69, 0x1d, 0xfc, 0x18 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the reception of segments with unaligned window scale option. Format 1. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.23 | 0x90cc4928, 0xd470, 0x491d, 0xaf, 0xa8, 0x9d, 0x86, 0x07, 0xb7, 0xf3, 0x15 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the reception of segments with unaligned window scale option. Format 2. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.24 | 0x5cd402e2, 0xe9d1, 0x40a7, 0x8a, 0xad, 0xe1, 0xc7, 0x89, 0x42, 0x52, 0x6b | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the reception of segments with unaligned window scale option. Format 3. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token*.Status* to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.25 | 0xe47378c6, 0x77d8, 0x4f08, 0xbb, 0x52, 0xe7, 0x6b, 0x14, 0xd9, 0x28, 0xd6 | EFI\_TCP4\_PROTOCOL **–** test when [OS]'s scaled window size larger than [OS]'s MSS, here, (256<<2) > 800, [EUT] could correctly send segment data with length small than MSS. With unaligned window scale option as format 2. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Check the Token**.Status** to verify the data has been transmitted successfully.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.26 | 0x82aacaa9, 0xa48e, 0x47c2, 0xb8, 0xa8, 0x88, 0xd3, 0x18, 0xf1, 0xd4, 0xe1 | EFI\_TCP4\_PROTOCOL **–** test TCP could disable timestamp option, when received <SYN> segment without timestamp while received data segment contain it. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. OS send DATA & ACK segment, then call EFI\_TCP4\_PROTOCOL.Receive()to receive a packet.  6. OS get the ACK segment and check the Token**.Status** to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.27 | 0xb0bf1171, 0x5e75, 0x42c4, 0x96, 0xed, 0x97, 0x21, 0xc6, 0x50, 0xe6, 0x87 | EFI\_TCP4\_PROTOCOL **–** test TCP could disable timestamp option, when it receives <SYN, ACK> segment without timestamp option. | 1. Build TCP Segment with MSS OPTION, MSS = 100.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  4. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Handle three-way handshake.  5. [OS] send SYN & ACK segment with MSS option and receive ACK segment.  6. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet. Then check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.28 | 0x6db78216, 0x1741, 0x4d22, 0x86, 0x2b, 0x1e, 0x37, 0x6f, 0x9f, 0xbe, 0xc9 | EFI\_TCP4\_PROTOCOL **–** test TCP could correctly recognize and deal with the timestamp option when it is used in TCP option. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. OS send DATA & ACK segment, then call EFI\_TCP4\_PROTOCOL.Receive()to receive a packet.  6. OS get the ACK segment and check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.29 | 0x688adc05, 0x942e, 0x4150, 0xa1, 0x6f, 0xec, 0xce, 0x9c, 0x3b, 0x66, 0x52 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the reception of segments with unaligned Timestamp option. Format 1. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. OS send DATA & ACK segment, then call EFI\_TCP4\_PROTOCOL.Receive()to receive a packet.  6. OS get the ACK segment and check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.30 | 0x98e5cf1f, 0x72ce, 0x4be6, 0x99, 0x95, 0x05, 0x43, 0xcd, 0x6c, 0x82, 0x93 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the reception of segments with unaligned Timestamp option. Format 2. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. OS send DATA & ACK segment, then call EFI\_TCP4\_PROTOCOL.Receive()to receive a packet.  6. OS get the ACK segment and check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.19.31 | 0x2f71233b, 0xeeaf, 0x4dc5, 0xb3, 0xdd, 0x35, 0x9f, 0xd6, 0xa6, 0xa2, 0x42 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly handles the reception of segments with unaligned Timestamp option. Format 3. | 1. Create TCP option. MSS = L\_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0.  2. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Handle three-way handshake.  5. OS send DATA & ACK segment, then call EFI\_TCP4\_PROTOCOL.Receive()to receive a packet.  6. OS get the ACK segment and check the Token*.Status* to verify the data has been transmitted successfully.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### Others

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.20.1 | 0xe78b5efa, 0xb455, 0x464e, 0xa2, 0x5f, 0xda, 0xf5, 0x3a, 0x14, 0x2c, 0x09 | EFI\_TCP4\_PROTOCOL **–**Tests that the [EUT] can correctly handle SYN flood. [EUT] should NOT send out <RST> segment to reset incomplete connection queue when ConnectionTimeout (SYN time) haven't reached. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  4. Send <SYN> flood, and wait to SYN timeout (ConncetionTimeout), then [EUT] send out <RST> segment to reset the incomplete connection.  5. Handles the three-way handshake. OS gets the <SYN, ACK> segment and then sends <ACK> segment.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.2 | 0x0c2a1607, 0xdff9, 0x4794, 0xb8, 0xca, 0x04, 0x28, 0x6a, 0xdf, 0xa8, 0x46 | EFI\_TCP4\_PROTOCOL **–**Tests that the [EUT] can correctly handle SYN flood. [EUT] accepts one or more connection request, thus making MaxSynBacklog NOT full. Accept following incoming <SYN> segment when MaxSynBacklog is NOT full. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  4. Send <SYN> flood.  5. Handles the three-way handshake. OS gets the <SYN, ACK> segment and then sends <ACK> segment.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.3 | 0xb8b111f9, 0xb3b7, 0x496b, 0x82, 0x5d, 0xaa, 0x9a, 0xd8, 0x59, 0x6c, 0x6e | EFI\_TCP4\_PROTOCOL **–**Tests that the [EUT] can correctly handle SYN flood. [EUT] should NOT send out <RST> segment to reset incomplete connection queue when ConnectionTimeout (SYN time) haven't reached. Discard following incoming <SYN> segment when MaxSynBacklog is full. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  4. Send <SYN> flood, and send <SYN> segment to [EUT] when MaxSynBacklog is full.  5. Handles the three-way handshake. OS gets the <SYN, ACK> segment.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.4 | 0x111f5b8e, 0xf762, 0x4eaf, 0x93, 0xb9, 0xe0, 0x97, 0xcb, 0x5b, 0xcd, 0x3f | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] can correctly handle attack-Self consume attack. | 1. Initialization of TCB related on OS side. Make the protocol address the same as [EUT], in order to attack.  2.Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  5. Send <SYN> flood.  6. Handles the three-way handshake. OS gets the <SYN, ACK> segment and sends <ACK> segment  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.5 | 0x8d7dd35a, 0x05f1, 0x495d, 0x8e, 0xed, 0x7e, 0x54, 0x70, 0x20, 0xd7, 0x67 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] can correctly handle attack-Self consume attack with SYN flood. | 1. Initialization of TCB related on OS side. Make the protocol address the same as [EUT], in order to attack.  2.Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  3. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  4. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance.  5. Handles the three-way handshake. OS gets the <SYN, ACK> segment and sends <ACK> segment  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.6 | 0xef277abd, 0xfe01, 0x4bbb, 0x91, 0x0d, 0xaa, 0xbb, 0x9f, 0x64, 0x68, 0xf4 | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality-Configure OS to send junky data after <FIN,ACK> segment, EUT should reset the connection. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handles the three-way handshake.  5. OS sends <FIN, ACK> segment and receives <ACK> segment.  6. OS sends DATA & ACK segment and then receives <RST, ACK> segment.  7. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.7 | 0xa1c11437, 0xbe91, 0x4857, 0x9e, 0xbc, 0x99, 0xfc, 0x3a, 0x3f, 0xba, 0x98 | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality-In CLOSE\_WAIT state, configure OS to send FIN to EUT. This FIN should not be duplicated of the last FIN segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake.  5. OS gets <SYN, ACK> segment and sends <ACK> segment. Then check the Token*.Status* to verify the EFI\_TCP4\_PROTOCOL.Accept() has completed.  6. OS sends <FIN, ACK> segment and receives <ACK> segment.  7. Calling EFI\_TCP4\_PROTOCOL.GetModeData(),now EUT is in CLOSE\_WAIT state.  8. OS sends <FIN, ACK> segment and receives <ACK> segment.  9. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.8 | 0xeb18fb2d, 0x2306, 0x41bc, 0x9d, 0x68, 0x10, 0x87, 0x8f, 0xf3, 0xe5, 0xef | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality-In LAST\_ACK state, configure OS to send FIN to EUT. This FIN should not be duplicated of the last FIN segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake.  5. OS gets <SYN, ACK> segment and sends <ACK> segment. Then check the Token*.Status* to verify the EFI\_TCP4\_PROTOCOL.Accept()has completed.  6. OS sends <FIN, ACK> segment and receives <ACK> segment.  7. Calling EFI\_TCP4\_PROTOCOL.GetModeData()**,** now EUT is in CLOSE\_WAIT state.  8. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow. Then call EFI\_TCP4\_PROTOCOL.GetModeData(), now EUT in LAST\_ACK state.  9. OS sends <FIN, ACK> segment and receives <ACK> segment.  10. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.9 | 0x968f5b4d, 0x4801, 0x487f, 0x81, 0xc1, 0xa6, 0x16, 0x91, 0x44, 0x47, 0x72 | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality-In TIME\_WAIT state, configure OS to send FIN to EUT. This FIN should not be duplicated of the last FIN segment. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Close() to do a graceful close working flow. Then call EFI\_TCP4\_PROTOCOL.GetModeData(), now EUT in FIN\_WAIT\_1 state.  5. OS sends <FIN, ACK> segment and receives <ACK> segment. Calling EFI\_TCP4\_PROTOCOL.GetModeData(), and now EUT is in TIME\_WAIT state. Then OS sends <FIN> segment.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.10 | 0x127d1f26, 0x9f39, 0x435c, 0x80, 0x34, 0x6b, 0x1c, 0xc9, 0x5e, 0x85, 0x3b | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality-Configure EUT to send data in no-ESTABLISHED state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet, without connection established.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.20.11 | 0x2d2065ef, 0x7e6a, 0x419a, 0x84, 0x30, 0x2c, 0x1d, 0xbf, 0xf7, 0x0c, 0xac | EFI\_TCP4\_PROTOCOL **–** Tests that the functionality-Configure EUT to send data in CLOSE\_WAIT state. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake. In addition, check the Token*.Status* to verify the EFI\_TCP4\_PROTOCOL.Accept() has completed.  4. OS sends <FIN, ACK> segment and receives <ACK> segment. Then call EFI\_TCP4\_PROTOCOL.GetModeData(), now EUT is in CLOSE\_WAIT state.  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to transmit a packet.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### KeepAliveTimer

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.21.1 | 0xece1fc13, 0x84f5, 0x413a, 0x90, 0xcb, 0x53, 0xfd, 0x45, 0x3a, 0x8d, 0x07 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly responds to the keep-alive segment which without garbage data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake.  4. Check [EUT] correctly responds to the keep-alive segment which without one garbage data.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.21.2 | 0x54e62a42, 0x25bb, 0x45c6, 0x90, 0x42, 0x93, 0x96, 0x8d, 0xab, 0xfc, 0x2c | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly responds to the keep-alive segment which with garbage data. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake.  4. Check [EUT] keeps connection when not all keep-alive probes were acknowledged.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### RetransmissiomTimer

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.22.1 | 0x64785c77, 0x4352, 0x4da5, 0xb0, 0xe8, 0x85, 0x0d, 0xdc, 0x5f, 0x32, 0x48 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly retransmit with the method of exponential back off. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.**Trasmit()** to make [EUT] send segment to [OS].  5. Call EFI\_TCP4\_PROTOCOL.Transmit() to check [EUT] correctly retransmit.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.22.2 | 0xf2474612, 0x61e6, 0x4bb9, 0x85, 0x7c, 0xb7, 0x00, 0x97, 0x05, 0x00, 0xdf | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly close connection when retransmission timer time out. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.**Trasmit()** to make [EUT] send segment to [OS].  5. Check [EUT] correctly performs retransmission timer.  6. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### HrdFormatACK

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.23.1 | 0xb550f0a9, 0x302a, 0x445a, 0x9b, 0xbf, 0xdb, 0xd3, 0x93, 0x9a, 0xec, 0x79 | EFI\_TCP4\_PROTOCOL **–**Tests that the [EUT] correctly generates the ACK numbers, and properly roll over the numbers. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Send Segment with seq 4294967294 to see EUTS whether return rollover ack.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |
| 5.25.1.23.2 | 0xced29cf0, 0xbfa9, 0x4b92, 0xb9, 0xe9, 0xdc, 0x3e, 0xc9, 0xea, 0x6a, 0x53 | EFI\_TCP4\_PROTOCOL **–**Tests that the [EUT] correctly generates the ACK numbers, and properly roll over the numbers. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake. Check the Token**.Status** to verify the connection has been established.  4. Send Segment with seq 4294967294 to see EUTS whether return rollover ack.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### HrdFormatCheckSum

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.24.1 | 0xeb8958d6, 0x9fac, 0x4c35, 0xa1, 0x66, 0xf2, 0x35, 0x1f, 0x43, 0x61, 0xb7 | EFI\_TCP4\_PROTOCOL **–**Test the [EUT]’s capability on generating a correct checksum field and discarding segments with invalid checksum. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as a passive one.  3. Call EFI\_TCP4\_PROTOCOL.Accept()for the new instance. Then handles the three-way handshake. Check the Token*.Status* to verify the connection has been established.  4. Send Segment with error CheckSum to see if EUTS discard this packet.  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

### PersistTimer

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.1.25.1 | 0xb498bbfe, 0xd47e, 0x4c9e, 0xb9, 0x80, 0x8f, 0x83, 0xc7, 0x33, 0xc6, 0x26 | EFI\_TCP4\_PROTOCOL **–** Tests that the [EUT] correctly performs persist timer with the method of exponential back off. | 1. Call EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Tcp4 child.  2. Call EFI\_TCP4\_PROTOCOL.Configure() to configure the new instance as an active one.  3. Call EFI\_TCP4\_PROTOCOL.Connect()for the new instance. Then handles the three-way handshake.  4. Call EFI\_TCP4\_PROTOCOL.**Trasmit()** to make [EUT] send segment to [OS].  5. Call **EFI\_TCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment. |

## EFI\_IP4\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_IP4\_PROTOCOL Section.

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.1.1 | 0xac92ef07, 0xd325, 0x4e3a, 0xad, 0x81, 0x46, 0x46, 0x3c, 0xb4, 0x0f, 0xa8 | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get all mode data when the Ip4 child has not been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.GetModeData() to get all mode data when the Ip4 child has not been configured. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.2 | 0x5abf337a, 0xfb74, 0x4812, 0x8c, 0xa3, 0x95, 0xb8, 0xbb, 0xed, 0x0b, 0xac | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get Ip4 mode data when the IP4 child has not been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.GetModeData() to get Ip4 mode data when the Ip4 child has not been configured. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.3 | 0x459937fd, 0x462d, 0x4b1f, 0x85, 0x78, 0x01, 0x78, 0xac, 0xcf, 0x2a, 0x2e | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get Mnp mode data when the IP4 child has not been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.GetModeData() to get Mnp mode data when the Ip4 child has not been configured. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.4 | 0x96463508, 0xc867, 0x410d, 0xab, 0x41, 0xc4, 0x3b, 0x54, 0x46, 0xe2, 0x53 | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get Snp mode data when the IP4 child has not been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.GetModeData() to get Snp mode data when the Ip4 child has not been configured. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.5 | 0x1b1253d6, 0xfb71, 0x4672, 0x84, 0xfa, 0xb4, 0x0a, 0x20, 0xb1, 0xc0, 0xae | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get all mode data when the IP4 child has been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get all mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.6 | 0xa27e3c75, 0xf51a, 0x4c22, 0x8c, 0x64, 0xb4, 0x52, 0xb9, 0xc6, 0xd6, 0xc6 | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get Ip4 mode data when the IP4 child has been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get Ip4 mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.7 | 0x0fa93b62, 0x3d3b, 0x40df, 0x8d, 0xea, 0x3f, 0x1e, 0x8e, 0xa2, 0x82, 0x1a | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get Mnp mode data when the IP4 child has been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get Mnp mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.8 | 0xefce9133, 0x49e6, 0x426c, 0x92, 0x38, 0x2a, 0x09, 0xda, 0x74, 0x30, 0x2d | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get Snp mode data when the IP4 child has been configured. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get Snp mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.9 | 0x6cbce077, 0x33b8, 0x4a73, 0x9e, 0x5a, 0x03, 0x41, 0xa9, 0xee, 0x44, 0xd4 | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get all mode data and check the *IcmpTypeList* data item. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get all mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS. Then check the *IcmpTypeCount* and *IcmpTypeList* data item.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.10 | 0x1fb8e582, 0x98c9, 0x461a, 0xbf, 0x26, 0xaf, 0x34, 0x6b, 0x1d, 0x23, 0xe0 | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get all mode data and check the *RouteTable*data item. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get all mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS. Then check the *RouteCount*and *RouteTable*data item.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.11 | 0x4f38bf49, 0x2be4, 0x489c, 0xac, 0xb9, 0x70, 0x3e, 0xb1, 0xe3, 0x5b, 0x3b | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get all mode data and check the *GroupTable*data item. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Groups() to add a group address.  3. Call EFI\_IP4\_PROTOCOL.GetModeData() to get all mode data when the Ip4 child has been configured. The return status should be EFI\_SUCCESS. Then check the *GroupCount*and *GroupTable*data item.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.1.12 | 0x3e8d5ff2, 0x5bec, 0x4e2d, 0xa6, 0x60, 0xe8, 0xfb, 0xe9, 0x8f, 0xb8, 0x49 | EFI\_IP4\_PROTOCOL.GetModeData() - invokes GetModeData() to check the instance status when Configure() has been called with an Ip4ModeData value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.GetModeData()and then check the IsStarted and IsConfigured item in Ip4ModeData.  4. Call EFI\_IP4\_PROTOCOL.Configure() with an Ip4ModeData value of NULL.  5. Call EFI\_IP4\_PROTOCOL.GetModeData(). The return status should be EFI\_SUCCESS. Then check the IsStarted and IsConfigured item in Ip4ModeData  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.2.1 | 0xf2e2bfe9, 0xe95d, 0x4c25, 0xa7, 0x0a, 0x59, 0x9c, 0xb7, 0x22, 0xcb, 0xde | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() with an StationAddressvalue of not an unicast IPv4 address. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance with an StationAddressvalue of not an unicast IPv4 address. The return status should be EFI\_INVALID\_PARAMETER.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.2 | 0x1c90fd78, 0x789d, 0x4710, 0x9b, 0x12, 0x27, 0xea, 0x09, 0xee, 0x99, 0x8b | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() with an SubnetMaskvalue of an invalid IPv4 subnet mask. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance with an SubnetMaskvalue of an invalid IPv4 address. The return status should be EFI\_INVALID\_PARAMETER.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.3 | 0x85e8e030, 0xf54a, 0x464c, 0x8e, 0xc7, 0xc8, 0xfb, 0x8f, 0x1a, 0x9b, 0xd1 | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() to change the StationAddress when the instance has been configured before. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Configure() again when the StationAddress has been changed. The return status should be **EFI\_ALREADY\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.4 | 0x62f11c24, 0xe8ff, 0x4687, 0x80, 0x3f, 0x40, 0x3f, 0x0f, 0x87, 0x0c, 0x8b | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() to change the SubnetMaskwhen the instance has been configured before. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Configure() again when the SubnetMaskhas been changed. The return status should be **EFI\_ALREADY\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.5 | 0xdddcb20e, 0x00a4, 0x4001, 0x85, 0x08, 0x60, 0x77, 0x3c, 0xfa, 0xba, 0xb8 | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() and call Transmit() and Receive() to check its function. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Transmit() to transmit a packet and check it is successful.  4. Call EFI\_IP4\_PROTOCOL.Receive() to receive the packet and check it is successful.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.6 | 0xdf081df1, 0x845a, 0x4ffe, 0x9a, 0xa3, 0x78, 0xc3, 0x77, 0xa1, 0x35, 0xc0 | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() and call Receive() to receive a packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Receive() to receive the packet and check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.7 | 0xedcd4582, 0x9349, 0x4f56, 0x9b, 0xac, 0x54, 0xe9, 0x2d, 0x6b, 0x27, 0xb4 | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() and call Receive() to receive a packet from different RemoteEther and RemoteIp. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Receive() to receive the packet from different *RemoteEther* and *RemoteIp*. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.8 | 0x90b93642, 0x81b3, 0x4d15, 0x9e, 0xbf, 0xdf, 0xc3, 0xaf, 0x70, 0xe1, 0xc6 | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() and call Transmit() to transmit a packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Transmit() to transmit the packet and check it is successful.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.2.9 | 0x171c383a, 0x613b, 0x4d85, 0x9c, 0xd4, 0x85, 0x57, 0x59, 0x4f, 0xb5, 0x67 | EFI\_IP4\_PROTOCOL.Configure() - invokes Configure() and call Transmit()andReceive() to check its function after call Configure() with an IpConfigData value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Configure() with an IpConfigData value of NULL. The return status should be EFI\_SUCCESS. Then call EFI\_IP4\_PROTOCOL.Transmit() and EFI\_IP4\_PROTOCOL.Receive()**,** the return status should be EFI\_NOT\_STARTED.  4. Call EFI\_IP4\_PROTOCOL.Configure() to configure the instance again. The return status should be EFI\_SUCCESS. Then call EFI\_IP4\_PROTOCOL.Transmit() and EFI\_IP4\_PROTOCOL.Receive()**,** the return status should be EFI\_SUCCESS.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Groups()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.3.1 | 0x360e7f0a, 0x635d, 0x4660, 0x95, 0x9c, 0x69, 0xa5, 0x39, 0x3c, 0x8d, 0x83 | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() with a JoinFlag value of TRUE and a GroupAddress value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() with a *JoinFlag* value of **TRUE** and a *GroupAddress* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.2 | 0x3ac80863, 0x67f2, 0x4554, 0x88, 0x72, 0xcd, 0x92, 0x98, 0xa1, 0xda, 0xac | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() with a GroupAddress value other than NULL and a *\**GroupAddressvalue of an invalid multicast IPv4 address. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() with a *GroupAddress* value other than NULL and a *\*GroupAddress*value of an invalid multicast IPv4 address. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.3 | 0x9634a43a, 0x41bc, 0x49f9, 0x80, 0x1c, 0x0e, 0xc1, 0x8b, 0xe1, 0x5c, 0x04 | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() to join a group address when it has already in the group table. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() to join a group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Groups() to join the group address again when it has already joined in step 3. The return status should be **EFI\_ALREADY\_STARTED**.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.4 | 0x4a2e6bd5, 0x2d4b, 0x4d81, 0xb5, 0x4b, 0x86, 0xc0, 0x03, 0x25, 0x9e, 0xf4 | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() to leave a group address which is not in the group table. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() to join a group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Groups() to leave the group address joined in step 3. The return status should be EFI\_SUCCESS.  5. Call EFI\_IP4\_PROTOCOL.Groups() to leave the group address again joined in step 3. The return status should be **EFI\_NOT\_FOUND**.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.5 | 0x1cc6a89f, 0xf635, 0x4aa6, 0xb2, 0x18, 0xfa, 0xc4, 0x7f, 0x7b, 0x83, 0x7c | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() when the instance has not been started. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Configure() again with an IpConfigData value of NULL.  4. Call EFI\_IP4\_PROTOCOL.Groups() with the a *JoinFlag* value of **TRUE** or **FALSE**. The return status should be **EFI\_NOT\_STARTED**.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.6 | 0x6138d5ae, 0x78b8, 0x43fa, 0x9a, 0x8c, 0x03, 0xb1, 0x87, 0x6d, 0x93, 0x15 | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() to join a group address and call Receive() to check that it is successful. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() to join a group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.**Receice()** to receive a packet from the group IP and check that it is successful.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.7 | 0x340a0020, 0x26ae, 0x4268, 0x87, 0x12, 0xe4, 0x58, 0x2d, 0x3e, 0x36, 0xe7 | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() to join two group address and call Receive() after leaving a group address from the group table. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() to join two group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Groups() to leave a group address from the group table.  5. Call EFI\_IP4\_PROTOCOL.**Receice()** to receive a packet from the group IP and check that it is successful.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.3.8 | 0x3234871f, 0x9682, 0x4bbd, 0x85, 0x56, 0x4a, 0x17, 0xa9, 0x74, 0xdf, 0xb7 | EFI\_IP4\_PROTOCOL.Groups() - invokes Groups() to leave all group address and call Receive() to check that it is successful. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Groups() to join two group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.**Receice()** to receive a packet from the group IP and check it can not receive the packet.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Routes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.4.1 | 0x9fa3288c, 0x1caa, 0x4174, 0xbc, 0x81, 0x84, 0x52, 0x16, 0x6f, 0x09, 0x58 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a DeleteRoute value of FALSE and a SubnetAddress value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a DeleteRoute value of FALSE and a SubnetAddress value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.2 | 0x6ed77fe8, 0xb20a, 0x417c, 0xb7, 0x64, 0x69, 0x36, 0x70, 0x74, 0xdf, 0x49 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a DeleteRoute value of FALSE and a SubnetMask value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a DeleteRoute value of FALSE and a SubnetMask value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.3 | 0x0ca07e01, 0xecf0, 0x4726, 0x8b, 0xb0, 0xb8, 0xd6, 0xde, 0xa2, 0x69, 0x77 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a DeleteRoute value of FALSE and a GatewayAddress value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a DeleteRoute value of FALSE and a GatewayAddress value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.4 | 0xe7ba143d, 0xb80c, 0x411b, 0xa7, 0xf7, 0x60, 0xa2, 0xb5, 0x10, 0xc7, 0x3d | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a DeleteRoute value of TRUE and a SubnetAddress value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a DeleteRoute value of TRUE and a SubnetAddress value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.5 | 0xf66dd341, 0xae38, 0x464e, 0x81, 0x22, 0x7f, 0xcb, 0xa4, 0x99, 0x1d, 0x31 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a DeleteRoute value of TRUE and a SubnetMask value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a DeleteRoute value of TRUE and a SubnetMask value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.6 | 0x713db4d5, 0x4e17, 0x487a, 0x83, 0x62, 0xe1, 0x18, 0x8b, 0x9f, 0x5e, 0x61 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a DeleteRoute value of TRUE and a GatewayAddress value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a DeleteRoute value of TRUE and a GatewayAddress value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.7 | 0xea35d39b, 0x7350, 0x427c, 0x8c, 0x04, 0x69, 0x0a, 0x75, 0x42, 0x75, 0x70 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a \*SubnetMask value of an invalid subnet mask. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a *\**SubnetMask value of an invalid subnet mask. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.8 | 0xe02b9e49, 0x3889, 0x4183, 0xac, 0x91, 0xb7, 0x4a, 0x63, 0xb5, 0xcf, 0x8f | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a \*GatewayAddress value of an invalid unicast IPv4 address. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a *\**GatewayAddress value of an invalid unicast IPv4 address. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.9 | 0x5a3132ea, 0x658e, 0x4bfb, 0xa3, 0xd2, 0x49, 0xeb, 0x6e, 0x88, 0xdf, 0xed | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() when the route has already been defined in the routing table (when DeleteRoute is FALSE). | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() to add a route into the routing table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Routes() to add the route again when it has already been defined in the routing table. The return status should be EFI\_ACCESS\_DENIED.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.10 | 0x5f228ffc, 0xfc1c, 0x43f6, 0x99, 0x14, 0x26, 0xcd, 0xcb, 0xee, 0x24, 0x97 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() to delete a route which is not in the routing table. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_PROTOCOL.Routes() to add a route into the routing table. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Routes() to delete the route added in step 3. The return status should be EFI\_SUCCESS.  5. Call EFI\_IP4\_PROTOCOL.Routes() to delete the route again while it is not in the routing table. The return status should be EFI\_NOT\_FOUND.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.11 | 0x3c71e7d7, 0xe61e, 0x4973, 0x90, 0xff, 0x36, 0x5b, 0xe5, 0xa7, 0x92, 0xb4 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() to add a route when using the default address and configuration has not finished yet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() to add a route into the routing table when using the default address and configuration has not finished yet. The return status should be EFI\_NO\_MAPPING.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.12 | 0xba7d5323, 0x36e4, 0x4b1a, 0x9e, 0x74, 0xdf, 0xe6, 0xd3, 0x30, 0xe5, 0xc5 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() delete a route when using the default address and configuration has not finished yet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() to delete a route into the routing table when using the default address and configuration has not finished yet. The return status should be EFI\_NO\_MAPPING.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.19 | 0xa51618f2, 0xe542, 0x4498, 0x82, 0xab, 0xc9, 0x9d, 0xc8, 0x61, 0x7f, 0xd0 | EFI\_IP4\_PROTOCOL.Routes() - Invoke Routes() when the driver instance has not been started. The return status should be EFI\_NOT\_STARTED. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Routes() to add a route into the routing table when the instance has not been started. The return status should be EFI\_NOT\_STARTED.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.13 | 0xf3239a4b, 0x29c1, 0x461e, 0xbf, 0x54, 0x96, 0x5d, 0xd9, 0x2e, 0x69, 0xb5 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a SubnetAddress value of "0.0.0.0",a SubnetMask value of "0.0.0.0" and a GatewayAddress value of "172.16.210.162". Then call Transmit() to check it. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a SubnetAddress value of "0.0.0.0",a SubnetMask value of "0.0.0.0" and a GatewayAddress value of "172.16.210.162". The return status should be EFI\_SUCCESS.  4. Call Ip.Transmit() to check the packet.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.14 | 0x7b17e47c, 0x0f7c, 0x4351, 0xa8, 0xfa, 0xf6, 0xf5, 0x9d, 0x03, 0x54, 0x93 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a SubnetAddress value of "172.16.210.0",a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "0.0.0.0". Then call Transmit() to check it. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a SubnetAddress value of "172.16.210.0",a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "0.0.0.0". The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Transmit() to check the packet.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.15 | 0x52762945, 0x2148, 0x48c9, 0x82, 0xea, 0xac, 0x78, 0xf3, 0x7c, 0xb7, 0x23 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() with a SubnetAddress value of "172.16.220.0", a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "172.16.210.162". Then call Transmit() to check it. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a SubnetAddress value of "172.16.220.0", a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "172.16.210.162". The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Transmit() to check the packet.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.16 | 0x91439045, 0x15f1, 0x4a25, 0x83, 0x0e, 0x4d, 0x0a, 0x2b, 0x2c, 0x13, 0x0a | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() to delete the route with a SubnetAddress value of "0.0.0.0", a SubnetMask value of "0.0.0.0" and a GatewayAddress value of "172.16.210.162". Then call Transmit() to check it. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a SubnetAddress value of "0.0.0.0", a SubnetMask value of "0.0.0.0" and a GatewayAddress value of "172.16.210.162". The return status should be EFI\_SUCCESS.  4. Call Ip.Routes() to delete the route added in step 3.  5. Call EFI\_IP4\_PROTOCOL.Transmit() to check the packet.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.17 | 0x3f884c4d, 0xcfd5, 0x49b8, 0x8f, 0x08, 0xfb, 0xb7, 0xb7, 0x44, 0x1e, 0xed | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() to delete the route with a SubnetAddress value of "172.16.210.0", a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "0.0.0.0". Then call Transmit() to check it. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a SubnetAddress value of "172.16.210.0", a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "0.0.0.0". The return status should be EFI\_SUCCESS.  4. Call Ip.Routes() to delete the route added in step 3.  5. Call EFI\_IP4\_PROTOCOL.Transmit() to check the packet.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.4.18 | 0x4745ddac, 0x9429, 0x4159, 0xbc, 0x13, 0x85, 0xf8, 0xd6, 0xe5, 0x23, 0x13 | EFI\_IP4\_PROTOCOL.Routes() - invokes Routes() to delete the route with a SubnetAddress value of "172.16.220.0", a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "172.16.210.162". Then call Transmit() to check it. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Routes() with a SubnetAddress value of "172.16.220.0", a SubnetMask value of "255.255.255.0" and a GatewayAddress value of "172.16.210.162". The return status should be EFI\_SUCCESS.  4. Call Ip.Routes() to delete the route added in step 3.  5. Call EFI\_IP4\_PROTOCOL.Transmit() to check the packet.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Transmit()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.5.1 | 0x47ba87f8, 0x188e, 0x4b41, 0x8d, 0x53, 0xa9, 0x08, 0x87, 0x73, 0x15, 0x6b | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit() with a Tokenvalue ofNULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.2 | 0x5701c82b, 0x64bf, 0x415e, 0x9f, 0x0f, 0x46, 0x23, 0x7b, 0x01, 0x91, 0xdf | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Event value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.3 | 0x44454955, 0x744c, 0x4648, 0xab, 0x05, 0x74, 0xac, 0x73, 0x0f, 0x9a, 0xa2 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxDatavalue ofNULL**.** | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxDatavalue ofNULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.4 | 0xf8e8550e, 0x46ff, 0x4e49, 0x81, 0xe5, 0xf7, 0x06, 0x5a, 0xd4, 0x84, 0xf9 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token*.Packet.OptionsLength* value other than 0 and a  Token.Packet.OptionsBuffer value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token*.Packet.OptionsLength* value other than 0 and a  Token*.Packet.OptionsBuffer* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.5 | 0x9edbcb93, 0xa28b, 0x40ed, 0x90, 0xfa, 0xa1, 0x7d, 0x41, 0xed, 0x93, 0x7d | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token*.Packet.FragmentCount*value of 0. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit() with a Token.Packet.FragmentCount value of 0. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.6 | 0x2ff682e3, 0x0b85, 0x4755, 0xaf, 0x58, 0x16, 0x57, 0x81, 0x23, 0x83, 0x2f | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with one or more of the Token.Packet.TxData*.*FragmentTable*[].FragmentLength* fields is 0. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with one or more of the Token.Packet.TxData*.*FragmentTable*[].FragmentLength* fields is 0. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.7 | 0x199e798a, 0x2f1a, 0x49ac, 0x81, 0x05, 0x91, 0xef, 0xc1, 0x24, 0x5b, 0xae | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with one or more of the Token.Packet.TxData*.*FragmentTable*[].FragmentBuffer*fields is NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with one or more of the Token.Packet.TxData*.*FragmentTable*[].FragmentBuffer*fields is NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.8 | 0x9bb3fb85, 0xbdfd, 0x4b0f, 0x95, 0x4c, 0x6a, 0x21, 0xbb, 0xff, 0x93, 0x7f | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxData*.*TotalDataLength value of 0. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxData*.*TotalDataLengthvalue of 0. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.9 | 0xff0221ac, 0x7a1c, 0x40e7, 0xbf, 0xea, 0xb2, 0xde, 0x89, 0xb2, 0xbf, 0x76 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxData*.*TotalDataLength not equal to the sum of fragment lengths. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxData*.*TotalDataLength not equal to the sum of fragment lengths. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.10 | 0xa22a64e0, 0xd98c, 0x49af, 0x98, 0xe1, 0x0d, 0x30, 0x93, 0x29, 0x7d, 0x34 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. (Set *SourceAddress* as "172.16.210.101" and GatewayAddress "172.16.210.255"). The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.11 | 0x2b27d386, 0xab2a, 0x4882, 0xa7, 0xf8, 0x71, 0xc0, 0xb6, 0x9c, 0xf9, 0x88 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. (Set *SourceAddress* as "172.16.210.101" and GatewayAddress "172.16.210.254"). The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.12 | 0x0251b68d, 0x32fe, 0x4b0e, 0xad, 0xe9, 0xc8, 0x45, 0x71, 0xd4, 0xfe, 0xec | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. (Set *SourceAddress* as "172.16.210.101" and GatewayAddress "240.0.0.2"). The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.13 | 0x3e687a19, 0x7b23, 0x45b7, 0x8f, 0x81, 0x0b, 0x1c, 0x28, 0xd5, 0x2a, 0x26 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()with a Token.Packet.TxData*.*OverrideData*.*GatewayAddressin the override data structure value of an invalid unicast IPv4 address if OverrideData is not NULL. (Set *SourceAddress* as "172.16.210.101" and GatewayAddress "255.255.255.255"). The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.14 | 0x00e45a87, 0xa739, 0x43af, 0xa7, 0x9f, 0x8d, 0xc7, 0xd3, 0x14, 0xab, 0x20 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit()when the IP header in FragmentTable is not a well-formed header when RawData is TRUE. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()when the IP header in FragmentTable is not a well-formed header when *RawData* is **TRUE**. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.15 | 0x4fc5e7c5, 0xdb04, 0x4d15, 0x94, 0xa4, 0x2d, 0xba, 0xac, 0x60, 0xbd, 0xbc | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit()when Token.Packet.TxData*.*TotalDataLength is not equal to the sum of fragment lengths. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()when Token.Packet.TxData*.*TotalDataLength is not equal to the sum of fragment lengths.(set Token.Packet.TxData*.*TotalDataLengthas 1). The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.16 | 0x5264d068, 0xe5a1, 0x41eb, 0x9d, 0x1e, 0xf8, 0xff, 0x20, 0x37, 0x77, 0x3a | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() when the length of the IPv4 header + option length + total data length is greater than the maximum packet size and DoNotFragment is TRUE. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()when the length of the IPv4 header + option length + total data length is greater than the maximum packet size and *DoNotFragment* is **TRUE**. The return status should be **EFI\_BAD\_BUFFER\_SIZE**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.17 | 0x383b9eb0, 0xb83a, 0x447d, 0x85, 0xcc, 0xd5, 0x2d, 0x49, 0xe5, 0x34, 0x8d | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() when the length of the IPv4 header + option length + total data length is greater than MTU. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()when the length of the IPv4 header + option length + total data length is greater than MTU. The return status should be **EFI\_BAD\_BUFFER\_SIZE**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.18 | 0x0ca2174b, 0x3731, 0x469f, 0x98, 0x2f, 0xb3, 0x45, 0xd8, 0xad, 0x7b, 0x4a | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() when the transmit completion token with the same Token.Event was already in the transmit queue. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet.  4. Call EFI\_IP4\_PROTOCOL.Transmit()with the same Token in step 2. The return status should be **EFI\_ACCESS\_DENIED**.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.31 | 0x45b5cb36, 0xf07a, 0x493c, 0xac, 0xee, 0x49, 0x91, 0x66, 0x6f, 0x0f, 0x00 | EFI\_IP4\_PROTOCOL.Transmit() - invoke Transmit() when the length of the IPv4 header + option length + total data length is greater than MTU.The return status shoule be EFI\_BAD\_BUFFER\_SIZE. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  4. Call EFI\_IP4\_PROTOCOL.Transmit()when the length of the IPv4 header + option length + total data length is greater than MTU. The return status shoule be **EFI\_BAD\_BUFFER\_SIZE**.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.19 | 0x394621bf, 0xe45c, 0x4dc7, 0x8c, 0x59, 0xa4, 0xb6, 0x25, 0xb0, 0x72, 0x4f | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() when there is no route found to destination address. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. Configure the IpConfigData *.*StationAddressnot same as TxData.DestinationAddress.  3. Call EFI\_IP4\_PROTOCOL.Transmit()when there is no route found to destination address. The return status should be **EFI\_NOT\_FOUND**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.20 | 0xb0e8dd55, 0x8e92, 0x4d9c, 0xba, 0x2d, 0x95, 0xcf, 0x35, 0x75, 0x71, 0x0b | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() when the instance has not been started. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. Then call EFI\_IP4\_PROTOCOL.Configure()again with aIpConfigData value of NULL**.**  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet. The return status should be **EFI\_NOT\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.32 | 0x3f38c35e, 0x92b8, 0x4e20, 0xaa, 0x23, 0x4b, 0xd9, 0xf6, 0xb3, 0x57, 0x7a | EFI\_IP4\_PROTOCOL.Transmit() - invoke Transmit() when the instance has not been started.The return status should be EFI\_NOT\_STARTED. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()when the instance has not been started.The return status should be **EFI\_NOT\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.21 | 0xac9ddcc1, 0xa095, 0x474b, 0x84, 0x06, 0x10, 0x37, 0xa4, 0x77, 0xe2, 0x24 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit an unicast packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit an unicast packet. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.22 | 0x3abee622, 0x0543, 0x46c6, 0xad, 0xfa, 0x97, 0x3a, 0x89, 0x6c, 0xbb, 0xdc | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a multicast packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a multicast packet. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.23 | 0xcc0ad3d9, 0xf1cd, 0x47e3, 0x81, 0x1d, 0xcb, 0x7a, 0x4e, 0x33, 0xd0, 0xfe | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a broadcast packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a broadcast packet. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.24 | 0x0979fc12, 0x53a1, 0x4cfb, 0x8c, 0xd7, 0xdf, 0xef, 0xb2, 0xc3, 0x76, 0x94 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet using OverrideData. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. Set IpConfigData *.*StationAddress "172.16.210.102” and IpConfigData *.*SubnetMask "255.255.255.0".  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet when set OverrideData*.SourceAddress* "172.16.210.101" and OverrideData*.*GatewayAddress "0.0.0.0". The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.25 | 0x3b0ae017, 0xcb82, 0x4f94, 0xb3, 0x17, 0xf7, 0x1d, 0x25, 0xe0, 0x33, 0xed | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet with TxData.OptionsLengthset as 4. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet with *TxData.OptionsLength*set as 4. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.26 | 0x2e24f6c8, 0x9fbf, 0x4fc3, 0xbb, 0x92, 0x1d, 0xd6, 0xab, 0xfa, 0xbd, 0x6f | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet with TxData.OptionsLengthset as 40 and initialize TxData.OptionsBuffer. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet with *TxData.OptionsLength*set as 40 and initialize *TxData.OptionsBuffer*. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.27 | 0x1da54ed7, 0x24d1, 0x4a19, 0xad, 0x19, 0x43, 0x89, 0x40, 0xd2, 0x73, 0xd2 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet with TxData.FragmentCountset as 4 and IpConfigData .DoNotFragment set as TRUE. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet with *TxData.FragmentCount*set as 4 and IpConfigData *.DoNotFragment* set as **TRUE**. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.28 | 0xbd451149, 0xc815, 0x4454, 0xb5, 0xf1, 0x8e, 0x14, 0x47, 0x6f, 0x91, 0x17 | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet with TxData.FragmentCountset as 4 and IpConfigData .DoNotFragment set as FALSE. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit 45 packets with *TxData.FragmentCount*set as 4 and IpConfigData *.DoNotFragment* set as **FALSE**. The return status should be EFI\_SUCCESS. Then check the captured packets number.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.29 | 0x298bc2eb, 0xa07b, 0x4e66, 0xba, 0xef, 0x2d, 0x03, 0x11, 0x72, 0xd4, 0xcb | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet with TxData.DestinationAddress set as "172.16.210.255" and FragmentTable.FragmentBuffer filled with char data. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet with *TxData.DestinationAddress* set as "172.16.210.255" and FragmentTable*.FragmentBuffer*filled with char data. The return status should be EFI\_SUCCESS. Then check packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.5.30 | 0x538a9496, 0x49a0, 0x4fe9, 0xa9, 0xe3, 0x0b, 0x20, 0x3f, 0xef, 0x03, 0xbb | EFI\_IP4\_PROTOCOL.Transmit() - invokes Transmit() to transmit a packet with FragmentTable.FragmentBufferfilled with UNIT8 data and FragmentTable.FragmentBuffer initialized. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Transmit()to transmit a packet when FragmentTable*.FragmentBuffer*filled with UNIT8 data and FragmentTable*.FragmentBuffer* initialized. The return status should be EFI\_SUCCESS. Then check packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Receive()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.6.1 | 0x31ee7913, 0x8cdf, 0x47dd , 0xa7, 0x29, 0xc9, 0x70, 0x51, 0xfc, 0x25, 0xfe | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive() with a Token value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Receive()to receive a packet with a Token value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.2 | 0x2ca314a9, 0x1afe, 0x40a3, 0xa4, 0x91, 0xc3, 0xe7, 0x2b, 0x02, 0x33, 0x7d | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive() with a Token.Event value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Receive()to receive a packet with a Token.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.3 | 0x4bb1005a, 0x5268, 0x4abf, 0x81, 0x34, 0x6d, 0x37, 0x0c, 0xde, 0x8e, 0x01 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()with the token that has already been placed in the receive queue. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance.  3. Call EFI\_IP4\_PROTOCOL.Receive() to receive a packet.  4. Call EFI\_IP4\_PROTOCOL.Receive()to receive a packet with the same Token.Event used in step 3. The return status should be **EFI\_ACCESS\_DENIED**.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.4 | 0xb9a3d3cd, 0xe982, 0x4268, 0xa7, 0x2a, 0xc3, 0xe5, 0xe8, 0xb6, 0xac, 0xa0 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()when the instance has not been started. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. Then call EFI\_IP4\_PROTOCOL.Configure()again withTokenisNULL**.**  3. Call EFI\_IP4\_PROTOCOL.Receive()to receive a packet. The return status should be **EFI\_NOT\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.5 | 0xf9658b87, 0x2377, 0x4fa2, 0xbe, 0x2a, 0x9c, 0x8d, 0x4b, 0x7e, 0xec, 0xe1 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()when an ICMP error packet was received. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Create (from IP head) and send an ICMP error packet, and Call EFI\_IP4\_PROTOCOL.Receive()to receive the packet. The return status should be EFI\_SUCCESS. Then check the packet field.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.6 | 0x134d695e, 0x6ea0, 0x46df, 0x8d, 0xbb, 0x62, 0x63, 0xf7, 0x1b, 0x29, 0x1a | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()when an ICMP error packet was received. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Create (from IP payload) and send an ICMP error packet, and Call EFI\_IP4\_PROTOCOL.Receive()to receive the packet. The return status should be EFI\_SUCCESS. Then check the packet field.  6. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.7 | 0x4aed29df, 0x95c0, 0x42b0, 0xaa, 0x65, 0xff, 0x72, 0xf1, 0x6d, 0x22, 0x4a | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive() to receive an ip packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Create an ip packet and call EFI\_IP4\_PROTOCOL.Receive()to receive the packet. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.8 | 0x47cb6918, 0xd454, 0x42f5, 0xa2, 0xab, 0x8e, 0xa5, 0x47, 0x3c, 0x6a, 0xab | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()to receive an ethernet packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Create an ethernet packet and call EFI\_IP4\_PROTOCOL.Receive()to receive the packet. The return status should be EFI\_SUCCESS. Then check the field of the packet.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.9 | 0xb2a56bae, 0x716d, 0x48b1, 0x9e, 0xc0, 0xd6, 0xbe, 0xed, 0xb2, 0x0e, 0xe2 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive() to receive 4 ip packets. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Create 4 ip packets and call EFI\_IP4\_PROTOCOL.Receive()to receive the packets. The return status should be EFI\_SUCCESS. Then check the packets field and count.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.10 | 0x452c7b90, 0xc99f, 0x4106, 0xbe, 0xce, 0x2d, 0xcd, 0x53, 0x50, 0x73, 0xd4 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()to receive 45 ip packets. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Create 45 ip packets and call EFI\_IP4\_PROTOCOL.Receive()to receive the packets. The return status should be EFI\_SUCCESS. Then check the packet field and count.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.11 | 0x5f497c40, 0xa1d3, 0x4223, 0xbc, 0x33, 0x4c, 0x8d, 0x96, 0x7d, 0xfc, 0xf7 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()to receive a broadcast ip packet. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.** SetIpConfigData .AcceptBroadcastisTRUE**.**  3. Create an ip packet and set RemoteEther FF:FF:FF:FF:FF:FF. call EFI\_IP4\_PROTOCOL.Receive()to receive the broadcast packet. The return status should be EFI\_SUCCESS. Then check the packet field.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.6.12 | 0x4be19438, 0xc5d8, 0x4af4, 0xaf, 0x0f, 0x8e, 0xc7, 0x49, 0x67, 0x2b, 0x40 | EFI\_IP4\_PROTOCOL.Receive() - invokes Receive()to receive an unformatted packet by set *RawData* with TRUE. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.** Set*RawData*with **TRUE.**  3. Create an ip packet and call EFI\_IP4\_PROTOCOL.Receive()to receive an unformatted packet. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Cancel()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.7.1 | 0x95d1ac2d, 0x4aaf, 0x4004, 0xb6, 0xa0, 0x8e, 0xec, 0x13, 0xd8, 0x31, 0xcc | EFI\_IP4\_PROTOCOL.Cancel() - invokes Cancel() when the asynchronous I/O request was not found in the transmit or receive queue. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Call EFI\_IP4\_PROTOCOL.Cancel() to abort an asynchronous transmit or receive request. The return status should be **EFI\_NOT\_FOUND**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.7.2 | 0xb41eab67, 0xc87c, 0x46a8, 0xae, 0x9d, 0x2c, 0xec, 0x34, 0xf7, 0x6d, 0x38 | EFI\_IP4\_PROTOCOL.Cancel() - invokes Cancel() with a Token value of NULL when the instance has not been started. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. Then call EFI\_IP4\_PROTOCOL.Configure() again with Token NULL.  3. Call EFI\_IP4\_PROTOCOL.Cancel() to abort an asynchronous transmit or receive request with aTokenvalue of NULL. The return status should be **EFI\_NOT\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.7.3 | 0x22fa385b, 0xc124, 0x41cd, 0xa6, 0xd9, 0x74, 0xf7, 0xc7, 0x78, 0x10, 0x88 | EFI\_IP4\_PROTOCOL.Cancel() - invokes Cancel() when the instance has not been started. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance. Then call EFI\_IP4\_PROTOCOL.Configure() again with Token NULL.  3. Call EFI\_IP4\_PROTOCOL.Cancel() to abort an asynchronous transmit or receive request. The return status should be **EFI\_NOT\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.7.4 | 0xd5bd141b, 0x5ade, 0x4831, 0xaf, 0x3c, 0x15, 0x46, 0xcd, 0xf4, 0xbc, 0x41 | EFI\_IP4\_PROTOCOL.Cancel() - invokes Cancel() to abort a receive request. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Call EFI\_IP4\_PROTOCOL.Receive() to receive a packet. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Cancel() to abort the asynchronous receive request. The return status should be EFI\_SUCCESS. Then check the status.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.7.5 | 0xf689d953, 0x1270, 0x448e, 0x93, 0xb1, 0xc0, 0xa5, 0x19, 0x1d, 0x6e, 0x10 | EFI\_IP4\_PROTOCOL.Cancel() - invokes Cancel()with a Token value of NULL to abort all receive requests. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.**  3. Call EFI\_IP4\_PROTOCOL.Receive() twice to put two receive requests. The return status should be EFI\_SUCCESS.  4. Call EFI\_IP4\_PROTOCOL.Cancel()with a Token value of NULL to abort all asynchronous receive requests. The return status should be EFI\_SUCCESS. Then check the status.  5. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### Poll()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.8.1 | 0x1c22cb9a, 0x14c5, 0x41a9, 0xa2, 0x00, 0x9e, 0x89, 0x90, 0xc4, 0x1b, 0xb4 | EFI\_IP4\_PROTOCOL.Poll() - invokes **Poll()** when the instance has not been started. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_PROTOCOL.Configure() to configure the new instance**.** Then callEFI\_IP4\_PROTOCOL.Configure()again withIpConfigData NULL**.**  3. Call EFI\_IP4\_PROTOCOL.**Poll()** for incoming data packets and processes outgoing data packets. The return status should be **EFI\_NOT\_STARTED**.  4. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.9.1 | 0xafda2aee, 0x1e1d, 0x4212, 0x82, 0x0a, 0x49, 0x69, 0x96, 0x8c, 0x26, 0xea | EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - invokes CreateChild() with a ChildHandle value of NULL. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child with a ChildHandlevalue ofNULL. the return status should be EFI\_INVALID\_PARAMETER.  2. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment. |
| 5.25.2.9.2 | 0x110c0779, 0x61f0, 0x46a5, 0x94, 0xd8, 0xe5, 0xf9, 0xfc, 0x24, 0xea, 0xba | EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – invokes CreateChild() to create several Ip4 childs. | Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create childs three times and then destroy them. |

### DestroyChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.2.10.1 | 0x7b89cc20, 0x3546, 0x4d7d, 0xae, 0x4b, 0xd7, 0xa6, 0xac, 0x94, 0xe9, 0x6b | EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - invokes DestroyChild() when the ChildHandle does not support the protocol that is removed. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() with the parameter ChildHandle that was created just now. the return status should be EFI\_SUCCESS.  3. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()with the parameter ChildHandle that was created just now. the return status should be **EFI\_UNSUPPORTED**. |
| 5.25.2.10.2 | 0x5e6fe618, 0x13a3, 0x4107, 0x8e, 0x1e, 0x35, 0xa8, 0x57, 0x84, 0x47, 0x12 | EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - invokes DestroyChild() to destroy a NULL child. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Ip4 child.  2. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() with the parameter ChildHandle isNULL.The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.2.10.3 | 0x08e3cc7b, 0x4441, 0x4bf3, 0xac, 0x61, 0xec, 0x2e, 0x63, 0x82, 0xb8, 0x17 | EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - invokes DestroyChild() to destroy the inexistent child. | 1. Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the inexistent child. The return status should be EFI\_INVALID\_PARAMETER**.** |
| 5.25.2.10.4 | 0x1400e3f9, 0x9681, 0x4da0, 0xbc, 0x18, 0xde, 0xce, 0xa8, 0x2f, 0x65, 0xf4 | EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - to test the function of DestroyChild(). | Call EFI\_IP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly three created Ip4 childs. |

## EFI\_IP4\_CONFIG\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_IP4\_CONFIG\_PROTOCOL Section.

### Start()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.3.1.1 | 0x5e97a936, 0xe3df, 0x4755, 0xa8, 0x33, 0x42, 0x4c, 0xd0, 0xd3, 0x38, 0xda | EFI\_IP4\_CONFIG\_PROTOCOL.Start() - invokes Start() when the parameter *DoneEvent* is NULL. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  2. Call EFI\_IP4\_CONFIG\_PROTOCOL**.Start()** to start the configuration process with a *DoneEvent* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  4. clean up the environment. |
| 5.25.3.1.2 | 0xe527172c, 0x26d9, 0x440a, 0x85, 0x4c, 0x15, 0x49, 0xfc, 0x6d, 0x5e, 0x49 | EFI\_IP4\_CONFIG\_PROTOCOL.Start() - invokes Start() when the parameter *ReconfigEvent* is NULL. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter DoneEvent.  2. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process with a *ReconfigEvent* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  4. clean up the environment. |
| 5.25.3.1.3 | 0xcd185521, 0xd395, 0x4be4, 0xbf, 0x0e, 0x21, 0x42, 0xc7, 0xb5, 0x1c, 0x78 | EFI\_IP4\_CONFIG\_PROTOCOL.Start() - invokes **Start()** when the configuration policy for the EFI IPv4 Protocol driver has already started. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process. The return status should be EFI\_SUCCESS.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process again. The return status should be **EFI\_ALREADY\_STARTED**.  5. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  8. clean up the environment. |
| 5.25.3.1.4 | 0x686babd0, 0x3be4, 0x4be1, 0x9a, 0xed, 0x38, 0x29, 0x83, 0x6a, 0xfc, 0x04 | EFI\_IP4\_CONFIG\_PROTOCOL.Start() - invokes Start() when the parameters *DoneEvent* and *ReconfigEvent* are not NULL. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process. The return status should be EFI\_SUCCESS.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  5. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  6. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  7. clean up the environment. |

### Stop()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.3.2.1 | 0xc5c3a59b, 0x4963, 0x43d5, 0x87, 0xfb, 0xc3, 0x53, 0x4c, 0x94, 0x5b, 0x38 | EFI\_IP4\_CONFIG\_PROTOCOL.Stop() - invokes **Stop()** when the configuration process has not been started. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process. The return status should be EFI\_SUCCESS.  5. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process again. The return status should be **EFI\_NOT\_STARTED**.  6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  8. clean up the environment. |
| 5.25.3.2.2 | 0x68d111a9, 0x35c6, 0x4e54, 0xaf, 0xae, 0x93, 0xc8, 0xe2, 0x95, 0xad, 0x3b | EFI\_IP4\_CONFIG\_PROTOCOL.Stop() - invokes **Stop()** to verify the configuration process. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process. The return status should be EFI\_SUCCESS.  5. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  6. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  7. clean up the environment. |

### GetData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.3.3.1 | 0xd21e8801, 0x7a1b, 0x4258, 0x84, 0xbe, 0x47, 0x68, 0xc0, 0x25, 0xe7, 0x1b | EFI\_IP4\_CONFIG\_PROTOCOL.GetData() - invokes **GetData()** when the configuration policy for the EFI IPv4 Protocol driver is not running. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  5. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  6. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  7. Call **EFI\_IP4\_CONFIG\_PROTOCOL.GetData()** to get configuration data when the driver is not running. The return status should be **EFI\_NOT\_STARTED**.  8. clean up the environment. |
| 5.25.3.3.2 | 0xb1b6d64a, 0xc963, 0x4d93, 0xaa, 0x56, 0xcd, 0xff, 0x2e, 0x09, 0x6a, 0x84 | EFI\_IP4\_CONFIG\_PROTOCOL.GetData() – invokes **GetData()** when EFI Ipv4 Protocol driver configuration is still running. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.GetData()** to get configuration data when the driver is still running. The return status should be **EFI\_NOT\_READY**.  5. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  8. clean up the environment. |
| 5.25.3.3.3 | 0x819d1861, 0xf092, 0x4c33, 0xbe, 0xf9, 0x8f, 0xf8, 0x8f, 0x05, 0xb2, 0xb3 | EFI\_IP4\_CONFIG\_PROTOCOL.GetData() – invokes **GetData()** when the parameter IpConfigData *Size* is smaller than the configuration data buffer. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.GetData()** to get configuration data with an IpConfigData *Size* value of 0. The return status should be **EFI\_BUFFER\_TOO\_SMALL**.  5. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  8. clean up the environment. |
| 5.25.3.3.4 | 0x1257612e, 0xe00c, 0x43d1, 0x97, 0xef, 0xfb, 0x60, 0x00, 0x30, 0x03, 0x1e | EFI\_IP4\_CONFIG\_PROTOCOL.GetData() – invokes GetData() when the parameter IpConfigData is NULL. | 1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  3. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.GetData()** to get configuration data with an IpConfigData value of NULL. The return status should be **EFI\_BUFFER\_TOO\_SMALL**.  5. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  8. clean up the environment. |
| 5.25.3.3.5 | 0x30710a44, 0x79e9, 0x45fc, 0x97, 0x4e, 0x3f, 0x48, 0x36, 0xbe, 0x33, 0xc8 | EFI\_IP4\_CONFIG\_PROTOCOL.GetData() – Test the function of GetData(). | 1. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to make sure configuration policy for the EFI IPv4 protocol driver is not running.  2. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*.  3. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*.  4. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Start()** to start the configuration process.  5. Send DHCPOFFER packet to agent.  6. Capture and validate DHCPREQUEST packet.  7. Send DHCPACK packet to agent  8. Call **EFI\_IP4\_CONFIG\_PROTOCOL.GetData()** to get configuration data.  9. Call **EFI\_IP4\_CONFIG\_PROTOCOL.Stop()** to stop the configuration process.  10. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*.  11. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*.  12. clean up the environment. |

## EFI\_TCP6\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_TCP6\_PROTOCOL Section.

### CreateChild()/DestroyChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.1.1 | 0xfca64cbc, 0xd99e, 0x42f0, 0x91, 0x23, 0x07, 0x76, 0xd7, 0x71,0x82, 0x9f | EFI\_TCP6\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_INVALID\_PARAMETER when ChildHandle isNULL. | Call CreateChild() when **ChildHandle** is **NULL**, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.1.2 | 0x991825b0, 0xd208, 0x429b, 0x98, 0xc9, 0x40, 0x46, 0xe5, 0x40, 0x00, 0x15 | EFI\_TCP6\_PROTOCOL.DestroyChild() - DestroyChild()returns EFI\_INVALID\_PARAMETER with ChildHandle beingNULL. | Call DestroyChild() when **ChildHandle** is **NULL**, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.1.3 | 0x7bfd1b83, 0x519b, 0x4bb4, 0x9a, 0x44, 0x12, 0x4a, 0xdc, 0x43, 0xdc, 0x56 | EFI\_TCP6\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_SUCCESS with valid parameters. | 5.25.4.1.3 to 5.25.4.1.6 belong to one case.  1. Call CreateChild() with valid parameters to create child1, The return status should be EFI\_SUCCESS. |
| 5.25.4.1.4 | 0x2d22615b, 0x8e8b, 0x44d2, 0x95, 0x25, 0xcc, 0x5c, 0x7e, 0x8c, 0x84, 0x54 | EFI\_TCP6\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_SUCCESS with valid parameters. | 2. Call CreateChild() with valid parameters to create child2, The return status should be EFI\_SUCCESS. |
| 5.25.4.1.5 | 0xd681c6b2, 0xa4d4, 0x4725, 0xab, 0xe5, 0xea, 0x5b, 0x03, 0x80, 0x76, 0xbf | EFI\_TCP6\_PROTOCOL.DestroyChild() - DestroyChild() returns EFI\_SUCCESS with valid parameters. | 3. Call DestroyChild() with valid parameters to destroy child1, The return status should be EFI\_SUCCESS. |
| 5.25.4.1.6 | 0x363eac60, 0x183a, 0x4b57, 0xae, 0x9e, 0x91, 0xcc, 0xf1, 0x95, 0x39, 0xfd | EFI\_TCP6\_PROTOCOL.DestroyChild() - DestroyChild() returns EFI\_SUCCESS with valid parameters. | 4. Call DestroyChild() with valid parameters to destroy child2, The return status should be EFI\_SUCCESS. |

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.2.1 | 0xd957c9de, 0x716a, 0x4f6e, 0xbe, 0x7c, 0x66, 0xc6, 0xe5, 0xa0, 0x2e, 0x09 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI NOT STARTED when the instance is not configured. | Call GetModeData() with valid parameters before the TCP instance is configured., the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.2.2 | 0x88a3650b, 0x3aa5, 0x4417, 0x97, 0x71, 0xef, 0xa4, 0xf6, 0xe5, 0x9a, 0x79 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 5.25.4.2.2 to 5.25.4.2.8 belong to one case.  1. Call GetModeData() with all no **NULL** input parameters, the return status should be EFI\_SUCCESS and the configured data should be correct. |
| 5.25.4.2.3 | 0x798259ad, 0xbc64, 0x4989, 0x9d, 0x8b, 0x82, 0x48, 0x01, 0x1a, 0x03, 0x06 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 2. Call GetModeData() with all **NULL** input parameters, the return status should be EFI\_SUCCESS. |
| 5.25.4.2.4 | 0xccb9b645, 0xf133, 0x4a2c, 0xbc, 0x72, 0xc1, 0xf1, 0xc8, 0x15, 0x05, 0xe5 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 3. Call GetModeData() when **TcpConnectionState** is **NULL**, the return status should be EFI\_SUCCESS. |
| 5.25.4.2.5 | 0xa9389312, 0x0007, 0x48ec, 0xab, 0x83, 0x26, 0x81, 0x1d, 0x0f, 0xa7, 0x97 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 4. Call GetModeData() when **TcpConfigData** is **NULL**, the return status should be EFI\_SUCCESS. |
| 5.25.4.2.6 | 0x8aa7bf92, 0xf01f, 0x4de8, 0x80, 0xab, 0x78, 0x9f, 0x4d, 0xaa, 0x16, 0x49 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 5. Call GetModeData() when **Ip6ModeData** is **NULL**, the return status should be EFI\_SUCCESS. |
| 5.25.4.2.7 | 0x92fcc066, 0xf41d, 0x4aad, 0xa6, 0x02, 0xf8, 0x4e, 0xde, 0x26, 0x15, 0x6d | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 6. Call GetModeData() when **MnpConfigData** is **NULL**, the return status should be EFI\_SUCCESS. |
| 5.25.4.2.8 | 0xb30b7510, 0x3055, 0x427d, 0x85, 0x4a, 0x79, 0xcd, 0xb1, 0xbb, 0xd2, 0x01 | EFI\_TCP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. | 7. Call GetModeData() when **SnpModeData** is **NULL**, the return status should be EFI\_SUCCESS. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.3.1 | 0xbebb71c0, 0xe62e, 0x400d, 0x9e, 0xaf, 0x3e, 0xbf, 0xb0, 0x23, 0xb2, 0xd6 | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_INVALID PARAMETERS when the station address is invalid. | Call Configure() when **StationAddress** is 2000::1(2000::1 is not configured for the testing environment), the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.3.2 | 0xabff27d2, 0x86ef, 0x4399, 0xbd, 0x90, 0x57, 0x8e, 0x8e, 0x08, 0x37, 0xb4 | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_INVALID PARAMETERS when the remote address is invalid. | Call Configure() when **RemoteAddress** is ff02::1(link local multicast address, not a valid unicast address), the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.3.3 | 0x1f16d3cc, 0x5ccf, 0x4177, 0x8b, 0xf2, 0x56, 0xde, 0x33, 0xe0, 0xd1, 0xf7 | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_INVALID PARAMETERS when the remote access point is invalid. | 5.25.4.3.3 to 5.25.4.3.4 belong to one case  1. Call Configure() when **RemoteAddress** is **::** and RemotePort is **8888**, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.3.4 | 0xae7a2155, 0x192e, 0x4bbb, 0x92, 0xc5, 0xad, 0x6d, 0x17, 0x57, 0xbc, 0xeb | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_INVALID PARAMETERS when the remote access point is invalid. | 2. Call Configure() when **RemoteAddress** is **2002::1** and RemotePort is **0**, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.3.5 | 0x3fea1f75, 0xce53, 0x4c85, 0xb8, 0xe5, 0x8e, 0x5a, 0x7c, 0x42, 0xeb, 0x64 | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_INVALID PARAMETERS when the access point has already been used by another instance. | 1. Create Child1 and call Configure() with valid parameters.  2. Create Child2 and call Configure() with the same access point. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.3.6 | 0xd8bc8edb, 0xfe65, 0x4457, 0xb5, 0x5a, 0xeb, 0xd4, 0xfa, 0xde, 0x7b, 0x7d | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_ACCESS DENIED when updating the configuration without reset. | 1. Call Configure() with valid parameters.  2. Call Configure() with valid parameters for the same instance. The return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.3.7 | 0xad816e3d, 0xf3e6, 0x443b, 0xa1, 0x54, 0x08, 0x51, 0xa5, 0x64, 0x63, 0xb4 | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_SUCCESS with valid parameters. | 5.25.4.3.7 to 5.25.4.3.8 belong to one case  1. Call Configure() with valid parameters. The return status should be EFI\_SUCCESS. |
| 5.25.4.3.8 | 0x85d67600, 0xf53b, 0x4363, 0x98, 0x34, 0xb9, 0x21, 0xaa, 0xf8, 0x8f, 0x08 | The Configure() should correctly set the data as expected. | 2. Call GetModeData() and check whether the data is set as expected. |
| 5.25.4.3.9 | 0x51b04624, 0xaa43, 0x4424, 0xa9, 0xb4, 0xee, 0x2f, 0x26, 0x24, 0xf5, 0x2f | The Tcp instance should enter into Tcp\_Listen state after being configured. | 5.25.4.3.9 to 5.25.4.3.13 belong to one case  1. Call Configure() with valid parameters.  2. Call GetModeData() to examine whether the Tcp\_ConnectionState is Tcp\_Listen. |
| 5.25.4.3.10 | 0x3d93a121, 0xde18, 0x4496, 0x87, 0xc2, 0xb7, 0x83, 0x0a, 0x92, 0xee, 0x0e | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_SUCCESS with valid parameters. | 3. Call Configure() when TcpConfigData is NULL. The instance should be reset correctly. |
| 5.25.4.3.11 | 0x9f6ad319, 0x0b1c, 0x40a0, 0x91, 0xee, 0xf9, 0x4e, 0x1a, 0xff, 0x9e, 0x09 | The Tcp instance should enter into Tcp\_Closed state after being reset. Call GetModeData() and the return value should be EFI NOT STARTED | 4. Call GetModeData(). The return value should be EFI NOT STARTED. |
| 5.25.4.3.12 | 0xea63c75a, 0x839f, 0x47b4, 0xad, 0x6c, 0x6f, 0xcf, 0x5f, 0xfd, 0x97, 0xfc | EFI\_TCP6\_PROTOCOL.Configure() – Configure() returns EFI\_SUCCESS with valid parameters. | 5. Call Configure() with valid parameters. |
| 5.25.4.3.13 | 0x0275b281, 0xf70e, 0x478d, 0xa6, 0x20, 0xa3, 0x28, 0x52, 0x5a, 0xd8, 0x07 | The Configure() should correctly set the data as expected. | 6. Call GetModeData() and check whether the data is set as expected. |

### Connect()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.4.1 | 0xa092e680, 0x27e9, 0x483b, 0xb3, 0xdb, 0x07, 0xb8, 0x69, 0x1a, 0xb7, 0xfc | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_NOT STARTED when the instance hasn’t been configured. | Call Connect()before the instance is configured, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.4.2 | 0x1e456f02, 0x7477, 0x4933, 0x84, 0xf9, 0x12, 0x9a, 0x8f, 0x64, 0x80, 0xa5 | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_INVALID PARAMETER when the token is NULL. | Call Connect()with the NULL token, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.4.3 | 0x3b5e2748, 0x1549, 0x465f, 0x98, 0x37, 0x67, 0xd9, 0x48, 0xdf, 0x50, 0x9f | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_INVALID PARAMETER when the token’s event is NULL. | Call Connect()when the token’s event is NULL, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.4.4 | 0x73f9316d, 0xbfcb, 0x4c3a, 0xbd, 0x75, 0x56, 0xb7, 0x03, 0x1d, 0x58, 0x30 | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_ACCESS DENIED when the instance is configured in passive mode. | 1. Call Configure()to configure the instance as passive mode.  2. Call Connect() with valid parameters, the return status should be **EFI\_ACCESS\_DENIED**. |
| 5.25.4.4.5 | 0xd15151a5, 0xf62b, 0x4203, 0x8e, 0x16, 0x47, 0x3b, 0x4a, 0x13, 0xd0, 0x89 | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_ACCESS DENIED when the instance is not in TCP\_CLOSED state. | 5.25.4.4.5 to 5.25.4.4.6 belong to one case  1. Call Configure()to configure the instance as active mode.  2. Call GetModeData() to check that the instance’s state should be TCP\_SYN\_SENT. |
| 5.25.4.4.6 | 0xf9de93e5, 0x4d4d, 0x45ab, 0x95, 0x0d, 0xc1, 0x53, 0x75, 0x51, 0xec, 0xb5 | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_ACCESS DENIED when the instance is not in TCP\_CLOSED state. | 3. Call Connect() when the instance’s state is not in TCP\_SYN\_SENT, The return status should be **EFI\_ACCESS\_DENIED**. |
| 5.25.4.4.7 | 0xfb14d45a, 0xa20d, 0x4c96, 0x94,0xc7, 0x86,0xc6, 0xc1,0x09, 0x9d,0xa4 | EFI\_TCP6\_PROTOCO  L.Connect() – Connect()  must return EFI\_CONNECTION\_REFUSED when the instance is  in SYN-RCVDstate & receive a RST | 1. Call  EFI\_TCP6\_SERVICE\_BINDING\_PROTO  COL.CreateChild() to create a new  Tcp6 child.  2. Call  EFI\_TCP6\_PROTOCOL.Configure()  to configure the new instance.  3. Call  EFI\_TCP6\_PROTOCOL.Connect()  Receive SYN & Send a SYN to put TCP state machine in SYN-RCVDstate.  4. Send a RST & check Connection Token state to be changed to EFI\_CONNECTION\_REFUSED  5. Call  EFI\_TCP6\_SERVICE\_BINDING\_PROTO  COL.DestroyChild() to destroy the created Tcp6 child and clean up the environment. |
| 5.25.4.4.8 | 0x3caf2371, 0x32e9, 0x4e29, 0x87, 0x64, 0x44, 0x12, 0x14, 0xcb, 0xa1, 0x63 | EFI\_TCP6\_PROTOCOL.Connect() - Connect()returns EFI\_SUCCESS with valid parameters. | 5.25.4.4.8 to 5.25.4.4.12 belong to one case  1. Call Connect() with valid parameters, the return status should be **EFI\_SUCCESS**. |
| 5.25.4.4.9 | 0xcd1704c9, 0xbabe, 0x4447, 0xaf, 0xda, 0xd2, 0x08, 0xc6, 0x9b, 0xd8, 0x8f | After the EFI\_TCP6\_PROTOCOL.Connect() is called, the EFI should send SYN packet successfully. | 2. Check whether the SYN packet is sent by SCT successfully. |
| 5.25.4.4.10 | 0x6e521181, 0x2a24, 0x4697, 0xbb, 0x83, 0x4b, 0xd9, 0xde, 0x5b, 0x89, 0xc0 | The TCP instance should acknowledge EMS’s SYN packet successfully. | 3. EMS send SYN packet to SCT side.  4. Check whether the ACK packet is sent by SCT successfully. |
| 5.25.4.4.11 | 0x1944bcf5, 0x9123, 0x469b, 0x86, 0xc2, 0x5c, 0x98, 0x7a, 0x39, 0xfe, 0x59 | The connection token’s event should be signaled successfully after 3-way handshakes are done. | 5. Check whether the token’s event is signaled after the 3-way handshake are done. |
| 5.25.4.4.12 | 0xcdae7179, 0xf66e, 0x4980, 0x9c, 0x08, 0x89, 0x0a, 0xe2, 0xcc, 0x4d, 0x46 | The connection token’s status should be modified to EFI\_SUCCESS after 3-way handshakes are done. | 6. Check whether the token’s status is modified as expected after the 3-way handshake are done. |

### Accept()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.5.1 | 0x30ec775a, 0xcefa, 0x4d56, 0x8c, 0x88, 0xa2, 0xdc, 0x75, 0x13, 0x56, 0x9c | EFI\_TCP6\_PROTOCOL.Accept() - Accept()returns EFI\_NOT STARTED when the instance hasn’t been configured. | Call Accept()before the instance is configured, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.5.2 | 0x08809174, 0x9447, 0x4956, 0x93, 0x0d, 0xa7, 0xb2, 0xa7, 0x63, 0x80, 0x9f | EFI\_TCP6\_PROTOCOL.Accept() - Accept()returns EFI\_ACCESS DENIED when the instance isn’t in passive mode. | Call Accept() with the instance in active mode, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.5.3 | 0x8f109af6, 0x55fe, 0x4f5c, 0x8b, 0x84, 0x22, 0xa8, 0x42, 0x4b, 0xc7, 0xf9 | EFI\_TCP6\_PROTOCOL.Accept() - Accept()returns EFI\_ACCESS DENIED when the listen token has already been queued. | 1. Call Accept()with valid parameters.  2. Call Accept()with the same token again, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.5.4 | 0xfc47ef2f, 0xc11c, 0x488c, 0x88, 0x21, 0xc8, 0xef, 0x3e, 0x2f, 0x3e, 0x7e | EFI\_TCP6\_PROTOCOL.Accept() - Accept()returns EFI\_INVALID PARAMETER when the listen token is NULL. | Call Accept() when the listen token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.5.5 | 0xf336471a, 0x6809, 0x4886, 0x95, 0x37, 0x2f, 0xf8, 0xb7, 0x5e, 0x5e, 0x8d | EFI\_TCP6\_PROTOCOL.Accept() - Accept()returns EFI\_INVALID PARAMETER when the event in the listen token is NULL. | Call Accept() when the event in the listen token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.5.6 | 0x19464085, 0x7ccc, 0x42a8, 0xbd, 0x81, 0x8a, 0x21, 0x0a, 0xf4, 0x70, 0xcd | EFI\_TCP6\_PROTOCOL.Accept() - Accept()returns EFI\_SUCCESS with valid parameters. | 5.25.4.5.6 to 5.25.4.5.14 belong to one case  1. Call Accept() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.4.5.7 | 0x2953f594, 0x8f06, 0x42f6, 0x8e, 0x7b, 0xc7, 0x8f, 0xf5, 0xc2, 0x4e, 0xa9 | The TCP instance should acknowledge EMS’s SYN packet successfully. | 2. EMS sent SYN packet to SCT side.  3. Check whether SCT accepts the SYN packet and send back SYN to EMS. |
| 5.25.4.5.8 | 0x04df3e6d, 0x599b, 0x43df, 0xb9, 0xb4, 0xf4, 0xaf, 0xc8, 0x3f, 0x48, 0x49 | The listen token’s event should be signaled successfully after 3-way handshakes are done. | 4. Check whether the token’s event is signaled after the 3-way handshake are done. |
| 5.25.4.5.9 | 0x727bb534, 0xd41f, 0x4132, 0x88, 0xbb, 0x8e, 0x02, 0xc6, 0x84, 0x2c, 0xbf | The listen token’s status should be modified to EFI\_SUCCESS after 3-way handshakes are done. | 5. Check whether the token’s status is modified as expected after the 3-way handshake are done. |
| 5.25.4.5.10 | 0xf88ff924, 0xfb1c, 0x4252, 0x9a, 0xa9, 0x18, 0xff, 0x46, 0xae, 0x75, 0x90 | The child handle contained in the listen token should not be NULL. | 6. Check whether the child handle contained in the token is NULL. |
| 5.25.4.5.11 | 0x1bff0f74, 0x465c, 0x4e25, 0xa6, 0x80, 0x8d, 0x2d, 0x43, 0x52, 0x28, 0x4d | The child handle contained in the listen token should be in TCP\_ESTABLISHED state. | 7. Check whether the child handle contained in the token is in correct state. |
| 5.25.4.5.112 | 0x06850748, 0xc64f, 0x4d44, 0xba, 0x43, 0x4e, 0xfb, 0xde, 0x2d, 0x2c, 0x7d | The child handle contained in the listen token should share the same configuration with its parent handle | 8. Check whether the child handle contained in the token has the same configuration as its parent handle. |
| 5.25.4.5.113 | 0x7415d9d3, 0x054f, 0x4a18, 0xb8, 0xbf, 0x6f, 0x6a, 0xae, 0xf4, 0xbc, 0x3f | Data communication should be correct on the child handle – Return value should be correct. | 9. Receive() with valid parameters, The return status should be EFI\_SUCCESS.  10. Check whether the event is signaled and the status is modified correctly. |
| 5.25.4.5.114 | 0x72834f64, 0x41fe, 0x46ab, 0x8b, 0x39, 0x64, 0xe3, 0x9f, 0x28, 0x6f, 0x71 | Data communication should be correct on the child handle – Data content should be as expected. | 11. Check whether the data length and data content for the Receive()is correct. |

### Transmit()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.6.1 | 0xef652675, 0x3d29, 0x4c9c, 0xbe, 0x90, 0xd3, 0xd6, 0x53, 0xac, 0x7b, 0x3c | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_NOT\_STARTED with the instance hasn’t been configured. | Call Transmit()before the instance is configured, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.6.2 | 0x31cbe783, 0xdea8, 0x4d05, 0x9b, 0x0b, 0xf0, 0x87, 0x5d, 0x3b, 0x07, 0x24 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_INVALID\_PARAMETER when the token is NULL. | Call Transmit() when the token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.6.3 | 0xcbb9c387, 0x96ef, 0x4834, 0xba, 0xeb, 0xe1, 0x9e, 0xca, 0x99, 0xae, 0xc7 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_INVALID PARAMETER when event in the token is NULL. | Call Transmit() when the event in token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.6.4 | 0xfdd4086f, 0xeffd, 0x4e7a, 0x93, 0xd2, 0x73, 0x74, 0x6d, 0x0f, 0x63, 0x18 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_INVALID PARAMETER when the TxData is NULL. | Call Transmit() when TxData is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.6.5 | 0xb3528e10, 0xd5ae, 0x4960, 0xb5, 0x03, 0xdd, 0x89, 0xd0, 0xf7, 0x6a, 0x09 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_INVALID PARAMETER when the FragmentCount is 0. | Call Transmit() when FragmentCount is 0, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.6.6 | 0xa8598edc, 0x469c, 0x4803, 0xbd, 0xf4, 0x37, 0xbf, 0x06, 0x8f, 0x41, 0x87 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_INVALID PARAMETER when the data length is not equal to the sum of all fragment buffers’ length. | Call Transmit() when the data length is not equal to the sum of all fragment buffers’ length, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.6.7 | 0x6231d7c6, 0xf61c, 0x4d6b, 0x94, 0xc4, 0xc6, 0xfc, 0x73, 0x59, 0xb6, 0xe2 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_ACCESS\_DENIED when the event has already been queued. | 1. Call Transmit() with valid parameters to send a data packet larger than MSS. The packet will be segmented to several bulks.  2. No ACK will be sent by EMS for the first segment. Hence, the event for the transmit token will stay in the queue.  3. Call Transmit() with the same event and valid other parameters again, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.6.8 | 0x5172270a, 0xf411, 0x4197, 0xbd, 0x34, 0x82, 0xc5, 0xc0, 0xe9, 0xa7, 0xcf | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_ACCESS\_DENIED when the instance has not been connected in active mode. | Call Transmit() in active mode before the 3-way handshake establishes, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.6.9 | 0x13fa7b6c, 0xdc0f, 0x4f9e, 0xae, 0x4a, 0x9e, 0x3e, 0x11, 0x02, 0xe2, 0x98 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_ACCESS\_DENIED when the instance has not been accepted in passive mode. | Call Transmit() in passive mode before the 3-way handshake establishes, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.6.10 | 0x9192cade, 0x7b3d, 0x44bf, 0x8a, 0xe7, 0x36, 0x28, 0x89, 0xd8, 0x76, 0x23 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_ACCESS\_DENIED when the instance has been closed. | Call Transmit() with valid parameters when the instance has been closed, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.6.11 | 0x8652c924, 0xf3d0, 0x43cc, 0x8b, 0xda, 0x8c, 0xd7, 0x16, 0xdc, 0xb3, 0xa0 | EFI\_TCP6\_PROTOCOL.Transmit() - Transmit()returns EFI\_SUCCESS with valid parameters. | 5.25.4.6.11 to 5.26.4.6.15 belong to one case  1. Call Transmit() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.4.6.12 | 0x096d60c6, 0xf036, 0x46be, 0xb0, 0xb2, 0x95, 0x13, 0xcf, 0xf1, 0x80, 0x81 | The transmitted packet should be delivered to network after the Transmit()is called. | 2. Check whether EMS could receive the transmitted packets in time. |
| 5.25.4.6.13 | 0x0d441d88, 0xd3eb, 0x4b97, 0x9c, 0x3d, 0xc9, 0xbe, 0xec, 0x2d, 0xeb, 0xc5 | The token event should be signaled after the packet is sent. | 3. Check whether the token event is signaled. |
| 5.25.4.6.14 | 0x9b0d226f, 0x4bc4, 0x4e1c, 0xb7, 0x07, 0xa1, 0x8e, 0x3a, 0x7b, 0x30, 0xf6 | The token status should be changed to EFI\_SUCCESS after the packet is sent. | 4. Check whether the token status is changed to EFI\_SUCCESS. |
| 5.25.4.6.15 | 0xfaca42a2, 0xa769, 0x4af9, 0x90, 0xcb, 0xf0, 0xd0, 0x5f, 0xf0, 0x8e, 0x03 | The packet length and content for the transmission should be correct. | 5. Check whether the packet length and content is correct. |

### Receive()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.7.1 | 0xd54cf9ed, 0x80e9, 0x44c0, 0x81, 0x25, 0xa7, 0x85, 0x2b, 0xbf, 0xec, 0x83 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_NOT\_STARTED when the instance hasn’t been configured. | Call Receive()before the instance is configured, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.7.2 | 0xa682e94a, 0x5d64, 0x4646, 0x98, 0x8d, 0x1e, 0x7a, 0xb1, 0x68, 0x8d, 0xb1 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_INVALID\_PARAMETER when the token is NULL. | Call Receive() when the token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.7.3 | 0xad9f6b64, 0xd0a0, 0x4bef, 0xbe, 0xdb, 0xf0, 0x42, 0x9b, 0x00, 0xfd, 0x76 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_INVALID PARAMETER when event in the token is NULL. | Call Receive() when the event in token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.7.4 | 0xc9a6cae7, 0x6e5e, 0x4c04, 0x9b, 0x1e, 0x27, 0xf3, 0x61, 0x34, 0x83, 0x8a | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_INVALID PARAMETER when the RxData is NULL. | Call Receive() when RxData is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.7.5 | 0x0cb365ff, 0xf855, 0x4ef5, 0xb8, 0xe5, 0xef, 0x2b, 0xc2, 0xd4, 0x6a, 0x7d | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_INVALID PARAMETER when the FragmentCount is 0. | Call Receive() when FragmentCount is 0, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.7.6 | 0x3ad62087, 0xfaf8, 0x4864, 0x9b, 0xd9, 0xad, 0xb1, 0x16, 0x6a, 0x54, 0x62 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_INVALID PARAMETER when the data length is not equal to the sum of all fragment buffers’ length. | Call Receive() when the data length is not equal to the sum of all fragment buffers’ length, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.7.7 | 0x4b325e98, 0x9ae8, 0x4a2b, 0x9e, 0x3e, 0x0a, 0xcf, 0x4a, 0x7e,0x69, 0x53 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_ACCESS\_DENIED when the event has already been queued. | 1. Call Receive() with valid parameters but no packet is sent from EMS. The receiving token will stay in the queue.  2. Call Receive() with the same event and other valid parameters again, The return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.7.8 | 0xddef303a, 0x3180, 0x466f, 0x80, 0x55, 0x26, 0xa4, 0x2f, 0x12, 0x1b, 0x78 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_ACCESS\_DENIED when the instance has not been connected in active mode. | Call Receive() in active mode before the 3-way handshake establishes, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.7.9 | 0x59b5cc95, 0xb0e9, 0x4cd6, 0xb1, 0x1d, 0x74, 0xcc, 0x26, 0x72, 0x33, 0x67 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_ACCESS\_DENIED when the instance has not been accepted in passive mode. | Call Receive() in passive mode before the 3-way handshake establishes, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.7.10 | 0xd985c3a0, 0xb98c, 0x4ad9, 0xb9, 0x9c, 0x1c, 0x5c, 0xfc, 0x4b, 0xea, 0xad | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_ACCESS\_DENIED when the instance has been closed. | Call Receive() with valid parameters when the instance has been closed, The return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.7.11 | 0xdcae30da, 0x090c, 0x441f, 0xbd, 0xa9, 0x02, 0x28, 0x4d, 0x2e, 0xab, 0xcb | EFI\_TCP6\_PROTOCO  L.Receive()– Receive() must return EFI\_CONNECTION\_FIN  .  When the communication peer has closed the connection and there is no any buffered data in the receive buffer of this instance | 1. Call  EFI\_TCP6\_SERVICE\_BINDING\_PROTO  COL.CreateChild()to create a new  Tcp6 child.  2. Call  EFI\_TCP6\_PROTOCOL.Configure()  to configure the new instance.  3. Call  EFI\_TCP6\_PROTOCOL.Connect()  & complete a 3-Way handshake    4. Send a FIN/ACK to close this connection  5.Call EFI\_TCP6\_SERVICE\_BINDING\_PROTO  COL.Receive()& check if its return status is EFI\_CONNECTION\_REFUSED  6. Call  EFI\_TCP6\_SERVICE\_BINDING\_PROTO  COL.DestroyChild() to destroy the created Tcp6 child and clean up the environment. |
| 5.25.4.7.12 | 0x2003bb96, 0xf32d, 0x48ca, 0x8e, 0x5a, 0x2c, 0x71, 0x6e, 0x95, 0x33, 0xf7 | EFI\_TCP6\_PROTOCOL.Receive() - Receive()returns EFI\_SUCCESS with valid parameters. | 5.25.4.7.12 to 5.26.4.7.15 belong to one case  1. Call Receive() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.4.7.13 | 0x5df1bf20, 0x8c5d, 0x4ef4, 0xb3, 0x70, 0xfd, 0x78, 0x14, 0xf2, 0x0a, 0x88 | The token event should be signaled after the packet is received. | 2. Check whether the token event is signaled. |
| 5.25.4.7.14 | 0xb65c6862, 0xebad, 0x4d51, 0xa1, 0xac, 0x73, 0xc0, 0x19, 0x24, 0x00, 0x8d | The token status should be changed to EFI\_SUCCESS after the packet is received. | 3. Check whether the token status is changed to EFI\_SUCCESS. |
| 5.25.4.7.15 | 0xfc18f3ec, 0xe779, 0x4730, 0x82, 0x24, 0xea, 0xdd, 0x9a, 0x4f, 0xd4, 0xf9 | The packet length and content for the received packet should be correct. | 4. Check whether the packet length and content is correct. |

### Close()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.4.8.1 | 0x97e34ed, 0x8b15, 0x479c, 0x9d, 0xa9, 0x57, 0x26, 0x58, 0x18, 0x72, 0x2d | EFI\_TCP6\_PROTOCOL.Close() - Close()returns EFI\_NOT\_STARTED with the instance hasn’t been configured. | Call Close()before the instance is configured, the return status should be EFI\_NOT\_STARTED. |
| 5.25.4.8.2 | 0x49ea02d4, 0x0022, 0x49c6, 0xac, 0x02, 0x3d, 0xe9, 0x96, 0x86, 0x48, 0xb9 | EFI\_TCP6\_PROTOCOL.Close() - Close()returns EFI\_INVALID\_PARAMETER when the token is NULL. | Call Close() when the token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.8.3 | 0x43dd8f75, 0x40d1, 0x4f54, 0x81, 0x5c, 0x81, 0x3e, 0xed, 0x71, 0x37, 0x89 | EFI\_TCP6\_PROTOCOL.Close() - Close()returns EFI\_INVALID PARAMETER when event in the token is NULL. | Call Close() when the event in token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.4.8.4 | 0xed7c5cd6, 0x0d5b, 0x4951,0xaa, 0x37, 0x96, 0xea, 0xe8, 0xa2, 0x7b, 0x89 | EFI\_TCP6\_PROTOCOL.Close() - Transmit()returns EFI\_ACCESS DENIED when the token event has already been used. | 1. Call Close() with valid parameters to perform a graceful close, but the EMS will send back no ACK. Hence the close event will stay in the queue.  2. Call Close() with the same event and valid other parameters, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.8.5 | 0x772e9c64, 0xc345, 0x4470, 0x9d, 0x93, 0x61, 0x71, 0xf8, 0x95, 0x52, 0x71 | EFI\_TCP6\_PROTOCOL.Close() - Transmit()returns EFI\_ACCESS DENIED when the last close has not been finished. | 1. Call Close() with valid parameters to perform a graceful close, but the EMS will send back no ACK. Hence the close event will stay in the queue and the first close will keep unfinished.  2. Call Close() with different event and valid other parameters, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.4.8.6 | 0x45385c8f, 0xa54a, 0x481d, 0xb2, 0x64, 0x3f, 0xc8, 0x12, 0xd1, 0x50, 0x39 | EFI\_TCP6\_PROTOCOL.Close() - Close()returns EFI\_SUCCESS with valid parameters. | 5.25.4.8.6 to 5.26.4.8.11 belong to one case  1. Call Close() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.4.8.7 | 0x764114c1, 0x2ba3, 0x4791, 0x96, 0x33, 0x35, 0xb2, 0x0b, 0x88, 0x43, 0xf4 | The FIN packet should be sent by SCT correctly. | 2. Check whether the FIN packet is sent out in time. |
| 5.25.4.8.8 | 0x10e12a40, 0x97c5, 0x467d, 0x97, 0x90, 0x0f, 0x58, 0x11, 0x84, 0xf1, 0x21 | The last ACK packet should be sent out correctly by SCT after receiving EMS’s FIN packet. | 3. After EMS receives the FIN packet. It sends out FIN/ACK packet to SCT.  4. Check whether the last ACK packet is sent out by SCT. |
| 5.25.4.8.9 | 0x333bdd81, 0x801d, 0x4aa1, 0x8c, 0x71, 0x31, 0x1d, 0x0f, 0x15, 0x89, 0x57 | The event in close token should be signaled. | 5. After the 4-way handshake finishes, check whether the close token’s event is signaled. |
| 5.25.4.8.10 | 0x33fa7b0c, 0x9e89, 0x4138, 0xa9, 0xaf, 0x3e, 0xee, 0x54, 0xa3, 0x90, 0x04 | The status of close token should be changed to EFI SUCCESS. | 6. Check whether the close token’s status is changed to EFI\_SUCCESS. |
| 5.25.4.8.11 | 0x1cdb5be1, 0xf8d0, 0x4570, 0x8e, 0x99, 0x7c, 0x6b, 0x6b, 0xb9, 0x76, 0x73 | The status of the TCP instance should be TCP\_CLOSEDafter the successful close(). | 7. Check whether the instance’s state is changed to TCP\_CLOSED. |
| 5.25.4.8.12 | 0x134177f3, 0x458a, 0x4088, 0x8e, 0x29, 0x84, 0x75, 0x1d, 0x68, 0x41, 0x43 | EFI\_TCP6\_PROTOCOL.Close() - Close()returns EFI\_SUCCESS with valid parameters when there is tokens in the queue. | 5.25.4.8.12 to 5.26.4.8.16 belong to one case  1. Transmit a large packet including several segments from SCT. EMS sends out ACK to the segments except for the last one. Hence the transmit token will pending in the queue.  2. Call Close() to close the connection, the return status should be EFI\_SUCCESS. |
| 5.25.4.8.13 | 0xb124b733, 0x1f2e, 0x4493, 0x95, 0xf6, 0x8e, 0xa3, 0x93, 0x1a, 0x8d, 0x6f | The FIN packet should be sent out immediately the last ACK is received. | 3. EMS sends out ACK for the last segment.  4. Check whether the SCT sends out FIN. |
| 5.25.4.8.14 | 0xede2639e,  0xa23b, 0x4ae5, 0xa0, 0xb3, 0x9d, 0x1c, 0x1b, 0x27, 0x90, 0x3d | The close token’s event should be signaled and status be changed correctly after the 4-way handshake finishes. | 5. EMS sends out FIN packet back to finish the 4-way handshake.  6. Check whether the close token’s event is signaled.  7. Check whether the close token’s status is changed to EFI\_SUCCESS. |
| 5.25.4.8.15 | 0x7c552532, 0x55ea, 0x46ac, 0x86, 0xf8, 0x0d, 0x1c, 0x27, 0x34, 0x71, 0xed | The TCP instance’s state should be TCP\_CLOSED after the 4-way handshake finishes. | 8. Check whether the instance’s state is changed to TCP\_CLOSED after the 4-way handshake finishes. |
| 5.25.4.8.16 | 0xdfe82050, 0x3325, 0x4dcf, 0xa0, 0xdc, 0xb7, 0x20, 0xa6, 0x72, 0xe9, 0xf0 | The pending transmit token should be signaled after the close finishes. | 9. Check whether the pending token is signaled or not. |
| 5.25.4.8.17 | 0x362144c2, 0xd822, 0x445a, 0x8d, 0x8d, 0x1a, 0x27, 0xcd, 0xf3, 0x17, 0x40 | EFI\_TCP6\_PROTOCOL.Close() - Close() to close and pending tokens should be signaled. | 1. Call Receive() to receive a incoming packet when there’s no packet sent from EMS. The receiving token will stay in the queue.  2. Call Close() to close the connection gracefully.  3. Check whether the receiving token is signaled and its state modified. |

## EFI\_IP6\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_IP6\_PROTOCOL Section.

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.1.1 | 0xc5a98289, 0xf32c, 0x4433, 0x81, 0xae, 0xa9, 0x10, 0xa3, 0x51, 0x0c, 0x32 | EFI\_IP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild()returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle. | Call CreateChild()with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.1.2 | 0x29d8f02c, 0xd19f, 0x48ec, 0xab, 0x8e, 0xb9, 0x10, 0x54, 0x10, 0x34, 0xc4 | EFI\_IP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – CreateChild()returns EFI\_SUCCESS with 1st valid ChildHandle. | 5.25.5.1.2 to 5.25.5.1.5 belong to one case  1. Call CreateChild()with the 1st  valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.25.5.1.3 | 0x3e7a34ce, 0x0a96, 0x4029, 0xa0, 0x0a, 0xd2, 0x7c, 0x75, 0x9c, 0xf0, 0x2d | EFI\_IP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – CreateChild()returns EFI\_SUCCESS with 2nd valid ChildHandle. | 2. Call CreateChild()with the 2nd valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.25.5.1.4 | 0x8e7bf890, 0x6109, 0x4d71, 0xa5, 0xb7, 0x83, 0x85, 0x0c, 0x5f, 0x78, 0x00 | EFI\_IP6\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() –DestroyChild()returns EFI\_SUCCESS with 2nd valid ChildHandle. | 3. Call DestroyChild()with the 2nd valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.25.5.1.5 | 0x974cd2fd, 0x79da, 0x4008, 0x92, 0x5a, 0x5c, 0x29, 0xa3, 0x7e, 0xd7, 0xb3 | EFI\_IP6\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() –DestroyChild()returns EFI\_SUCCESS with 1st valid ChildHandle. | 3. Call DestroyChild()with the 1st valid ChildHandle, the return status should be EFI\_SUCCESS. |

### DestoryChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.2.1 | 0x5b7d1b2f, 0x41f1, 0x4787, 0xa6, 0xb5, 0xfa, 0x28, 0x9e, 0x34, 0xcd, 0xd3 | EFI\_IP6\_SERVICE\_BINDING\_PROTOCOL.DestoryChild() - DestoryChild() returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle. | Call DestoryChild()with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.3.1 | 0xc8a6f564, 0x2320, 0x46fa, 0xbf, 0x2a, 0x0b, 0x77, 0x3c, 0x71, 0x1d, 0xf6 | EFI\_IP6\_PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters | 5.25.5.3.1 to 5.25.5.3.2 belong to one case  1. Call GetModeData() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.3.2 | 0x3919816b, 0xf3bd, 0x4177, 0x8d, 0x90, 0xf3, 0xca, 0xba, 0x20, 0x9a, 0xc2 | Validate the IP6ModeData.IsConfigured | 2. The value of IP6ModeData.IsConfigured should be FALSE. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.4.1 | 0x99fe5cde, 0xdccb, 0x4d55, 0xab, 0xb4, 0xa1, 0xdf, 0x73, 0x30, 0x2d, 0x4b | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.StationAddress is neither zero nor a valid unicast Ipv6 address. | Call Configure()when IpConfigData.StationAddress is neither zero nor a valid unicast Ipv6 address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.5.4.2 | 0xa0998aa3, 0x7f5e, 0x401f, 0x8f, 0x3d, 0xeb, 0xe9, 0x09, 0x5c, 0xbd, 0x7b | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.StationAddress is neither zero nor one of configured Ipv6 address. | Call Configure()when Ip6ConfigData.StationAddress is neither zero nor one of configured Ipv6 address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.5.4.3 | 0xafca1a79, 0xc38f, 0x4e5a, 0x8b, 0xa9, 0x33, 0xaf, 0xd9, 0x04, 0x7b, 0xbf | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.4 | 0xcc598692, 0xc3e7, 0x4008, 0x91, 0xc2, 0x29, 0xf6, 0xc4, 0x0f, 0x74, 0x41 | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData*.*DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.5 | 0x6aa9538e, 0x3e88, 0x4309, 0xab, 0x52, 0x94, 0xc5, 0x09, 0x3e, 0x9a, 0x34 | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.6 | 0x84a3a2cb, 0x3bc5, 0x47f9, 0xab, 0xb4, 0xd5, 0xa6, 0x89, 0xfa, 0x1a, 0x80 | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when *Ip6ConfigData.DefaultProtocol* is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.7 | 0x43804768, 0xca58, 0x4f59, 0xa8, 0x18, 0x1b, 0x0e, 0x9a, 0x0f, 0xc1, 0xa6 | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.8 | 0xecfe10f7, 0xce1f, 0x4711, 0xb0, 0xc8, 0xd8, 0x56, 0xe5, 0x35, 0x4a, 0x82 | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when *Ip6ConfigData.DefaultProtocol* is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.9 | 0xa9c4db07, 0x17f3, 0x43e3, 0xa7, 0x43, 0x78, 0xe9, 0x51, 0xb7, 0x35, 0xce | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.10 | 0x64e2f4e1, 0x4431, 0x490a, 0xa0, 0x2f, 0xe3, 0xb4, 0x0c, 0x80, 0x12, 0xbb | EFI\_IP6 PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.11 | 0x1224d773, 0x44fb, 0x44db, 0xba, 0xb5, 0x63, 0x75, 0x5d, 0x11, 0x20, 0xdb | EFI\_IP6\_PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER when Ip6ConfigData.DefaultProtocol is invalid. | Call Configure()when Ip6ConfigData.DefaultProtocol is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.4.12 | 0xf380d0c6, 0x2b60, 0x4674, 0xa8, 0xec, 0x94, 0x8c, 0x21, 0xbd, 0xc7, 0xd7 | EFI\_IP6\_PROTOCOL.Configure() - Configure()returns EFI\_ALREADY\_STARTED with valid Ip6ConfigData which isn't NULL but the instance has been configured. | Call Configure()with valid Ip6ConfigData which isn't NULL when the instance has been configured, the returns status should be EFI\_ALREADY\_STARTED. |
| 5.25.5.4.13 | 0x217fe9de, 0x908c, 0x4eb8, 0xac, 0xaa, 0x74, 0x96, 0x23, 0xf5, 0x25, 0x98 | EFI\_IP6\_PROTOCOL.Configure() – Configure() returns EFI\_SUCCESS with valid parameters. | 5.25.5.4.13 to 5.25.5.4.16 belong to one case.  1. Call Configure()with valid parameters; the returns status should be EFI\_SUCCESS. |
| 5.25.5.4.14 | 0xc53003dd, 0xd76d, 0x47ca, 0xae, 0x09, 0x1a, 0xed, 0x49, 0x00, 0xc6, 0x9c | EFI\_IP6\_PROTOCOL.GetModeData() – GetModeData() returns EFI\_SUCCESS with valid parameters. | 2. Call GetModeData()with valid parameters after the child configured, the returns status should be EFI\_ SUCCESS. |
| 5.25.5.4.15 | 0x48f68c63, 0x4860, 0x4993, 0x8f, 0xc2, 0x1b, 0x73, 0x28, 0x21, 0xcb, 0x22 | Validate the IP6ModeData.ConfigData. | 3. Validate the IP6ModeData.ConfigData.The IP6ModeData.ConfigData should be the same as the data which have been configured before. The returns status should be EFI\_SUCCESS. |
| 5.25.5.4.16 | 0x8287365d, 0x46e5, 0x406b, 0x98, 0x2c, 0x75, 0xdc, 0x39, 0x99, 0xd7, 0x5b | Validate the IP6ModeData.IsConfiged. | 4. Call Configure()with NULL and then Call GetModeData() with valid parameters, and validate the IP6ModeData.IsConfiged*.* It should beFALSE. |

### Groups()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.5.1 | 0x756d489b, 0x1d6d, 0x4ab5, 0x99, 0x72, 0xd1, 0x96, 0x4a, 0x7b, 0x28, 0x0f | EFI\_IP6\_PROTOCOL.Groups() - Groups() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Groups()with a not configured ChildHandle; the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.5.2 | 0x2c1abd64, 0x7657, 0x4f78, 0x9f, 0x2c, 0xfa, 0x48, 0xf2, 0xd7, 0xbb, 0x66 | EFI\_IP6\_PROTOCOL.Groups() - Groups() returns EFI\_INVALID\_PARAMETER when JoinFlag isTRUE and GroupAddress isNULL | Call Groups() when JoinFlag is TRUE andGroupAddress isNULL.The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.5.3 | 0x6053a2b7, 0x391a, 0x4b46, 0xa7, 0x34, 0x1e, 0x2e, 0x86, 0x5c, 0x39, 0x82 | EFI\_IP6\_PROTOCOL.Groups() - Groups() returns EFI\_INVALID\_PARAMETER when GroupAddress is not NULL and GroupAddress is not a multicast IPv6 address. | Call Groups()when GroupAddress is not NULL and GroupAddress is not a multicast IPv6 address. The returned status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.5.4 | 0x1644ec0d, 0x4ef0, 0x42b8, 0xad, 0x6b, 0x8b, 0xbd, 0xd5, 0x3f, 0x84, 0x1d | EFI\_IP6\_PROTOCOL.Groups() - Groups() returns EFI\_ALREADY\_STARTED when JoinFlag is TRUE and GroupAddress is in the group table. | Call Groups()when JoinFlag is TRUE and GroupAddress is in the group table, the return status should be EFI\_ALREADY\_STARTED. |
| 5.25.5.5.5 | 0xc1fe68df, 0xca52, 0x42c4, 0xbe, 0xd4, 0xc0, 0x34, 0xf9, 0xf0, 0x03, 0x18 | EFI\_IP6\_PROTOCOL.Groups() - Groups() returns EFI\_NOT\_FOUND when JoinFlag is FALSE and GroupAddress is not in the group table. | Call Groups()when JoinFlag is FALSE and GroupAddress is not in the group table, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.5.6 | 0xbf971751, 0xbc7e, 0x421a, 0x86, 0xbe, 0xda, 0x67, 0x16, 0x03, 0xb0, 0xf0 | EFI\_IP6\_PROTOCOL.Groups() - Groups() returns EFI\_SUCCESS with TRUE JoinFlag and an valid GroupAddress. | 5.25.5.5.6 to 5.25.5.5.10 belong to one case.  1. Call Groups()with TRUE JoinFlag and a valid GroupAddress, the return status should be EFI\_SUCCESS. |
| 5.25.5.5.7 | 0x3542d69e, 0xc8eb, 0x4da6, 0x8e, 0x41, 0xdd, 0x49, 0x43, 0x17, 0xa7, 0x80 | Check the Ip6ModeData.GroupCount field. | 2. The value of Ip6ModeData.GroupCount should be 1. |
| 5.25.5.5.8 | 0x65dafab8, 0xe505, 0x4f4a, 0xa7, 0xaf, 0x54, 0x42, 0x68, 0x42, 0xca, 0xa8 | Check the Ip6ModeData.GroupTable field. | 3. The value of Ip6ModeData.GroupTable should be the same as the route entry we added. |
| 5.25.5.5.9 | 0x25af1861, 0x25e5, 0x4137, 0xb1, 0xb0, 0x56, 0x5f, 0xfa, 0x32, 0xee, 0x44 | EFI\_IP6\_PROTOCOL.Groups () - Groups() returns EFI\_SUCCESS with FALSE JoinFlag and and GroupAddress is in the group table. | 4. Call Groups()with FALSE JoinFlag and and GroupAddress is in the group table, the return status should be EFI\_SUCCESS. |
| 5.25.5.5.10 | 0x882ddbc2, 0x4372, 0x41ff, 0x95, 0x5c, 0x89, 0x15, 0x56, 0x73, 0xb3, 0x5d | Check the Ip6ModeData.GroupCount field. | 5. Call GetModeData() with valid parameters, the value of Ip6ModeData.GroupCount should be 0. |

### Routes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.6.1 | 0xe5a50efc, 0x831b, 0x4dc1, 0x8a, 0x78, 0xb5, 0x36, 0xa2, 0x39, 0xd8, 0x8d | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Routes ()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.6.2 | 0x9a9fadb0, 0x6651, 0x4070, 0xac, 0x63, 0x2b, 0xa0, 0x92, 0xc5, 0xe0, 0x0b | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_INVALID\_PARAMETER when DeleteRoute is TRUE, both Destiniation and GatewayAddress are NULL. | Call Routes()when DeleteRoute is TRUE, both Destiniation and GatewayAddress are NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.6.3 | 0x38dabbd5, 0x37fb, 0x4744, 0xab, 0x18, 0xac, 0xcf, 0x5d, 0x0e, 0x25, 0xf1 | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_INVALID\_PARAMETER when DeleteRoute is FALSE, Destiniation is NULL and GatewayAddress is not NULL. | Call Routes() when *DeleteRoute* is FALSE, Destiniation is NULL and GatewayAddress is not NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.6.4 | 0xb3ea5648, 0x9a8c, 0x4761, 0x9f, 0x9c, 0x9b, 0x44, 0x87, 0xca, 0x14, 0x0a | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_INVALID\_PARAMETER when DeleteRoute is FALSE, *Destiniation* is not NULL and GatewayAddress is NULL. | Call Routes() when *DeleteRoute* is FALSE, Destiniation is not NULL and GatewayAddress is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.6.5 | 0xef4878ab, 0x02e1, 0x4a3f, 0x9b, 0x0c, 0x0a, 0xea, 0x7d, 0x25, 0xf2, 0x46 | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_INVALID\_PARAMETER when GatewayAddress is not a valid unicast IPv6 address. | Call Routes()when GatewayAddress is not a valid unicast IPv6 address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.6.6 | 0x67ab6941, 0xfe7d, 0x4046, 0x9f, 0xc4, 0x61, 0x6c, 0x50, 0xb9, 0xd3, 0x72 | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_INVALID\_PARAMETER when GatewayAddress is one of configured local IPv6 addresses. | Call Routes()when GatewayAddress is one of configured local IPv6 addresses, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.6.7 | 0x2359c3c5, 0x5789, 0x4c12, 0xbc, 0x1c, 0x5b, 0x94, 0x18, 0x5d, 0x24, 0x39 | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_NOT\_FOUND when DeleteRoute is TRUEand this entry is not in current routing table. | Call Routes()when DeleteRoute is TRUEand this entry is not in current routing table, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.6.8 | 0x9c9e4191, 0xbd67, 0x42d7, 0x8e, 0x64, 0x22, 0xe4, 0xc3, 0x4b, 0x8c, 0x2e | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_ACCESS\_DENIED when **DeleteRoute** is FALSE and the entry is already in current routing table. | Call Routes()when DeleteRoute is FALSE and the entry is already in current routing table, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.5.6.9 | 0x576be5b1, 0xc50e, 0x44d3, 0x80, 0x99, 0xa0, 0x67, 0x56, 0x0b, 0x24, 0x10 | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_SUCCESS with valid parameters. | 5.25.5.6.9 to 5.25.5.6.13 belong to one case.  1. Call Routes()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.6.10 | 0x8c3d2c17, 0xc282, 0x4daa, 0x96, 0xfb, 0x1d, 0x1c, 0xdc, 0xd2, 0x9f, 0x99 | Check Ip6ModeData.RouteCount field | 2. The value of Ip6ModeData.RouteCount should more than zero. |
| 5.25.5.6.11 | 0xb7cc7815, 0x7a38, 0x4904, 0xb2, 0x4d, 0x22, 0x09, 0x00, 0xb5, 0xf7, 0xcc | Check Ip6ModeData.RouteTable field*.* | 3. Ip6ModeData.RouteTableshould contain the route we added before. |
| 5.25.5.6.12 | 0x709e8127, 0x1a36, 0x4c08, 0xac, 0x22, 0xd1, 0xb5, 0x0f, 0x82, 0x5a, 0x14 | EFI\_IP6\_PROTOCOL.Routes() - Routes() returns EFI\_SUCCESS with valid parameter . | 4. Call Routes()with valid parameters to delete the route we added before, the return status should be EFI\_SUCCESS. |
| 5.25.5.6.13 | 0xe30d8352, 0x4f0c, 0x43fe, 0xb2, 0x0e, 0xcf, 0xeb, 0xfb, 0x45, 0xb4, 0x42 | CheckIp6ModeData.RouteCountfield*.* | 5. The value of Ip6ModeData.RouteCountshould be decreased by 1. |

### Neighbors()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.7.1 | 0x4f6a49b0, 0xff4f, 0x4ba8, 0xa6, 0x31, 0x94, 0x8d, 0x23, 0xbc, 0x15, 0x00 | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_NOT\_STARTED with a not configured ChildHandle*.* | Call Neighbors()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED*.* |
| 5.25.5.7.2 | 0x35ffe726, 0x0b87, 0x480e, 0xa2, 0xeb, 0x1c, 0x7d, 0xed, 0x16, 0x99, 0x4e | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_INVALID\_PARAMETER when TargetIp6Address is NULL*.* | Call Neighbors()when TargetIp6Address is NULL, the return status should be EFI\_INVALID\_PARAMETER*.* |
| 5.25.5.7.3 | 0x3360d9f1, 0x674a, 0x445f, 0xab, 0x8a, 0x3b, 0xca, 0xde, 0xae, 0xed, 0x2b | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_INVALID\_PARAMETER when TargetLinkAddress is NULL and DeleteFlag is TRUE*.* | Call Neighbors()when TargetLinkAddress is NULL and DeleteFlag is TRUE, the return status should be EFI\_INVALID\_PARAMETER*.* |
| 5.25.5.7.4 | 0xc0556979, 0x5ab6, 0x4c65, 0xb6, 0x49, 0xc7, 0xbe, 0x34, 0x9f, 0x04, 0xed | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_INVALID\_PARAMETER when TargetLinkAddress is invalid. | Call Neighbors()when TargetLinkAddress is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.7.5 | 0x98c0eda5, 0xf1b5, 0x4bf3, 0xa1, 0x58, 0xbb, 0x68, 0xdc, 0xe3, 0xb4, 0x5c | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_INVALID\_PARAMETER when TargetIpAddress is not a valid unicast Ipv6 Address. | Call Neighbors()when TargetIpAddress is not a valid unicast Ipv6 Address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.7.6 | 0xe60636fa, 0x47f1, 0x433e, 0xa0, 0x79, 0x50, 0x92, 0xcf, 0x59, 0x0b, 0xb1 | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_INVALID\_PARAMETER when TargetIpAddress is one of configured local Ipv6 address. | Call Neighbors()when TargetIpAddress is one of configured local Ipv6 address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.7.7 | 0xd88a65be, 0x37ff, 0x41e2, 0xa8, 0xbd, 0x3e, 0x92, 0x1b, 0xf5, 0x89, 0x87 | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_NOT\_FOUND when DeleteFlag is **TRUE** and this entry isn't in current neighbor cache. | Call Neighbors()when DeleteFlag is **TRUE** and this entry isn't in current neighbor cache, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.7.8 | 0x7a528a8e, 0x1339, 0x4618, 0x92, 0x9e, 0xf5, 0x60, 0xb6, 0xd1, 0x98, 0xd0 | EFI\_IP6 PROTOCOL.Neighbors() - Neighbors() returns EFI\_ACCESS\_DENIED when DeleteFlag is **FALSE** and this entry isn't in current neighbor cache. | Call Neighbors()when DeleteFlag is **FALSE** and this entry isn't in current neighbor cache, the return status should be EFI\_ ACCESS\_DENIED. |
| 5.25.5.7.9 | 0xb0c66678, 0x6552, 0x42f7, 0xa4, 0x5a, 0x36, 0x3d, 0xde, 0xa5, 0x75, 0xbd | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_NOT\_FOUND when DeleteFlag is **FALSE** and the **TargetLinkAddress** is **NULL**. | Call Neighbors()when **DeleteFlag** is **FALSE** and the TargetLinkAddress is **NULL**, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.7.10 | 0xf339086f, 0xd826, 0x48b4, 0xbf, 0x77, 0xd7, 0x71, 0xba, 0xb6, 0x28, 0xb5 | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_SUCCESS with valid parameters. | 5.25.5.7.10 to 5.25.5.7.15 belong to one case  1. Call Neighbors()with valid parameters to add a neighbor cache, the return status should be EFI\_SUCCESS. |
| 5.25.5.7.11 | 0xa5389777, 0xd3d2, 0x41da, 0xa7, 0x22, 0xbf, 0xbe, 0xe2, 0xc8, 0x78, 0x4e | Check Ip6ModeData.NeighborCount field. | 2. The value of Ip6ModeData.NeighborCount should be 1. |
| 5.25.5.7.12 | 0x179fa1e4, 0xa408, 0x481d, 0xbb, 0x3a, 0x72, 0x81, 0x2e, 0xcd, 0x2a, 0xde | Check Ip6ModeData.NeighborsCache.Neighbor field. | 3. The value of Ip6ModeData.NeighborsCache.Neighbor should be the same as we added. |
| 5.25.5.7.13 | 0x6991227c, 0x3562, 0x4875, 0x82, 0x2e, 0x7d, 0xe3, 0xf3, 0xcf, 0x90, 0x59 | Check Ip6ModeData.NeighborsCache.LinkAddressfield. | 4. The value of Ip6ModeData.NeighborsCache.LinkAddress should be the same as we added. |
| 5.25.5.7.14 | 0x823ca277, 0xdaa3, 0x4917, 0xa2, 0x58, 0xc9, 0xe3, 0x30, 0xef, 0xb6, 0xd1 | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_SUCCESS with valid parameters. | 5. Call Neighbors()with valid parameters to delete a neighbor cache, the return status should be EFI\_SUCCESS. |
| 5.25.5.7.15 | 0x971bf190, 0x49c5, 0x4b5b, 0x83, 0x20, 0x0c, 0x74, 0xc3, 0x5c, 0xc9, 0x91 | Check Ip6ModeData.NeighborCount field. | 6. The value of Ip6ModeData.NeighborCount should be 0 after delete. |
| 5.25.5.7.16 | 0x0379e4c1, 0x2b4f, 0x41e2, 0xb6, 0x44, 0xda, 0xf5, 0x4a, 0x53, 0xd9, 0xdd | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_SUCCESS with valid parameters. | 5.25.5.7.16 to 5.25.5.7.22 belong to one case  1. Call Neighbors()with valid parameters to add a neighbor cache, the return status should be EFI\_SUCCESS. |
| 5.25.5.7.17 | 0xeb7f4f6f, 0x521e, 0x452c, 0xbc, 0x6e, 0xdf, 0xbf, 0xb9, 0x22, 0x2e, 0x3b | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_SUCCESS with valid parameters. | 2. Call Neighbors()with valid parameters to update a neighbor cache, the return status should be EFI\_SUCCESS. |
| 5.25.5.7.18 | 0x53567ad3, 0x2cfe, 0x4bfd, 0xba, 0x97, 0xea, 0xca, 0xad, 0xdd, 0x2d, 0x00 | Check Ip6ModeData.NeighborCountfield. | 3. The value of Ip6ModeData.NeighborCount should be 1 after added**.** |
| 5.25.5.7.19 | 0x6be12cd9, 0xcdf7, 0x4b0c, 0x82, 0xb5, 0x5b, 0xee, 0x3c, 0xfd, 0x52, 0xe8 | Check Ip6ModeData.NeighborsCache.Neighbor field. | 4. The value of Ip6ModeData.NeighborsCache.Neighbor should be the same as we added. |
| 5.25.5.7.20 | 0x8dfbc45e, 0x5b6d, 0x4c1d, 0x9c, 0x0a, 0x2f, 0xcc, 0xb6, 0x1e, 0xeb, 0xfa | Check Ip6ModeData.NeighborsCache.LinkAddress field. | 5. The value of Ip6ModeData.NeighborsCache.LinkAddress should be the same as we added. |
| 5.25.5.7.21 | 0xe9aa5a6e, 0x9b98, 0x4e3d, 0xa2, 0xc1, 0x49, 0x31, 0x85, 0x14, 0x72, 0xde | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_SUCCESS with valid parameters. | 6. Call Neighbors()with valid parameters to delete a neighbor cache, the return status should be EFI\_SUCCESS. |
| 5.25.5.7.22 | 0x2d82ca70, 0xc383, 0x458e, 0x93, 0x1d, 0x84, 0xfd, 0x2b, 0xb2, 0x7c, 0xfd | Check Ip6ModeData.NeighborCount field. | 7. The value of Ip6ModeData.NeighborCount should be 0 after deleted**.** |
| 5.25.5.7.23 | 0x5646fc4f, 0x06cb, 0x49ba, 0xbe, 0xb0, 0x3d, 0xf0, 0xde, 0x02, 0xda, 0xbf | EFI\_IP6\_PROTOCOL.Neighbors() - Neighbors() returns EFI\_SUCCESS with valid parameters. | 5.25.5.7.23 to 5.25.5.7.27 belong to the same case  1. Call Neighbors()with valid parameters to add a neighbor cache, the return status should be EFI\_SUCCESS. |
| 5.25.5.7.24 | 0x4baa627a, 0x0019, 0x4eda, 0xbd, 0x27, 0xbb, 0xd2, 0xdd, 0x5f, 0x9f, 0x19 | Check Ip6ModeData.NeighborCount field. | 2. The value of Ip6ModeData.NeighborCountshould be 1 after added**.** |
| 5.25.5.7.25 | 0xa93cf6a1, 0x3548, 0x41e8, 0x94, 0xdc, 0x07, 0xe8, 0x30, 0x72, 0x34, 0xd5 | Check Ip6ModeData.NeighborsCache.Neighbor field. | The value of Ip6ModeData.NeighborsCache.Neighbor should be the same as we added. |
| 5.25.5.7.26 | 0xe0297637, 0x7b3d, 0x4894, 0x80, 0x8d, 0x2c, 0x7d, 0x64, 0xa9, 0x19, 0x46 | Check Ip6ModeData.NeighborsCache.LinkAddress field. | The value of Ip6ModeData.NeighborsCache.LinkAddress should be the same as we added. |
| 5.25.5.7.27 | 0xa03dc0e3, 0xffe3, 0x4bff, 0x82, 0x9f, 0xb0, 0x99, 0xb3, 0xe2, 0x57, 0x64 | Check Ip6ModeData.NeighborCount field. | The value of Ip6ModeData.NeighborCount should be 0 after time out**.** |

### Transmit()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.8.1 | 0x255fe450, 0xc537, 0x4b0a, 0xbe, 0x80, 0xc8, 0x73, 0x95, 0x66, 0x26, 0x16 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Transmit()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.8.2 | 0x8347ebcd, 0x4f16, 0x4bfd, 0x83, 0xf6, 0x0f, 0x8a, 0xdc, 0x6a, 0x89, 0x2e | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token. | Call Transmit()with a NULL Token, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.3 | 0xc7cf4815, 0x9c64, 0x4074, 0x94, 0x3f, 0xf5, 0x6d, 0x2e, 0x9d, 0x79, 0x5d | EFI\_IP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Event. | Call Transmit()with a NULL Token->Event, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.4 | 0x2ccfe480, 0x452c, 0x4706, 0x88, 0x69, 0x97, 0xb7, 0x7b, 0x03, 0xa9, 0x26 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Packet.TxData. | Call Transmit()with a NULL Token->Packet.TxData, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.5 | 0xede110b2, 0x8455, 0x4ec8, 0xbb, 0x22, 0x19, 0x94, 0x59, 0x54, 0x11, 0x46 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER when Token->Packet.TxData->ExtHdrs is NULL. | Call Transmit()when Token->Packet.TxData->ExtHdrs is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.6 | 0xd4f4a746, 0xaff3, 0x4490, 0xa6, 0xd9, 0xef, 0x38, 0x06, 0x69, 0x0a, 0x94 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER when Token->Packet.TxData->FragmentCount is Zero. | Call Transmit()when Token->Packet.TxData->FragmentCount is Zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.7 | 0xa2dc1ca1, 0x37ef, 0x4147, 0xa6, 0x90, 0x4d, 0x4e, 0xd1, 0x4c, 0x99, 0xf9 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER when Token->Packet.TxData->FragmentTable[0].FragmentLength is Zero. | Call Transmit()when **T**oken->Packet.TxData->FragmentTable[0].FragmentLength is Zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.8 | 0xef828012, 0xdeda, 0x4f91, 0xb1, 0x10, 0x38, 0x26, 0x92, 0x50, 0xf3, 0xc8 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Packet.TxData->FragmentTable[0].FragmentBuffer. | Call Transmit()with a NULL Token->Packet.TxData->FragmentTable[0].FragmentBuffer, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.9 | 0x8db7ffb3, 0x47fb, 0x4281, 0x97, 0xa5, 0x8a, 0xa7, 0xe1, 0x98, 0x87, 0x72 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER Token->Packet.TxData->DataLength is zero. | Call Transmit()when Token->Packet.TxData->DataLength is zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.10 | 0x63c9939b, 0x7aa6, 0x4565, 0xab, 0x11, 0xdc, 0x13, 0x32, 0x38, 0x1b, 0x32 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with an invalid Token->Packet.TxData->DataLength which is not equal to the sum of fragments length. | Call Transmit()with an invalid Token->Packet.TxData->DataLength which is not equal to the sum of the fragments length, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.11 | 0x220f2e8c, 0xae0c, 0x4f9c, 0x89, 0x1b, 0x74, 0x54, 0xaa, 0x63, 0xf0, 0xce | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a non-zero Token->Packet.TxData->Udp6sessionData->DestinationAddress which is not specified in configure process. | Call Transmit()with a non-zero Token->Packet.TxData->Udp6sessionData->DestinationAddress which is not specified in configure process, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.12 | 0xc7353218, 0xc96e, 0x4236, 0x92, 0x53, 0x86, 0x85, 0x41, 0x0a, 0x47, 0x0c | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a zero Token->Packet.TxData->Udp6sessionData->DestinationAddress when DestinationAddressis unspecified when doing configure process. | Call Transmit()with a zero Token->Packet.TxData->Udp6sessionData->DestinationAddress when DestinationAddressis unspecified when doing configure process, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.8.13 | 0x2ac52cba, 0xbe4e, 0x4c9e, 0xae, 0xe5, 0x4d, 0x10, 0x6b, 0x95, 0x1b, 0xc4 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_ACCESS\_DENIED with a Token->Event which has already been in the transmit queue. | Call Transmit()with a Token->Event which has already been in the transmit queue, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.5.8.14 | 0xfeaa4963, 0x24c0, 0x477a, 0x8a, 0xc7, 0xa9, 0xac, 0xe5, 0xbb, 0xf4, 0x53 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_NOT\_FOUND with no route entry to the destination. | Call Transmit()with no route entry for the destination, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.8.15 | 0xda08e7a1, 0x7ab6, 0x4b23, 0x9b, 0xb6, 0x27, 0xae, 0x0a, 0xb7, 0xb6, 0xc3 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_BAD\_BUFFER\_SIZE with a Token->Packet.TxData->DataLength which beyond the maximum udp6 packet size. | Call Transmit()with a Token->Packet.TxData->DataLength which beyond the maximum udp6 packet size, the return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.5.8.17 | 0x4660050c, 0x749c, 0x428f, 0xa5, 0xd9, 0x9a, 0x4c, 0x8e, 0xa4, 0x20, 0xe5 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_SUCCESS with valid parameters. | 5.25.5.8.17 to 5.25.5.8.21 belong to one case.  1. Call Transmit()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.8.18 | 0xb67c0483, 0x7b89, 0x446c, 0xac, 0xba, 0x17, 0xb8, 0x7f, 0x4e, 0xcb, 0x5f | Token->Event should be signaled. | 2. Token->Event should be signaled. |
| 5.25.5.8.19 | 0x9a61d143, 0x7ddf, 0x4d4e, 0xa7, 0x97, 0x5f, 0xfc, 0x85, 0x09, 0x0e, 0xb4 | Token->Status should be EFI\_SUCCESS. | 3. Token->Status should be EFI\_SUCCESS. |
| 5.25.5.8.20 | 0x8916816a, 0x6876, 0x4e76, 0xa2, 0xc2, 0x3d, 0xc6,0x3f, 0xcd, 0x00, 0x7a | The packet should be received by the other side. | 4. The packet should be received by the other side. |
| 5.25.5.8.21 | 0x088ed948, 0x0276, 0x4bb4, 0x98, 0x96, 0xe3, 0xa7, 0x67, 0x21, 0x74, 0x2f | The received packet content should be reasonable. | 5. The received packet content should be reasonable. |
| 5.25.5.8.22 | 0x3cf5b8eb, 0xc742, 0x4d34, 0x97, 0x65, 0xf8, 0xcc, 0x32, 0x49, 0x4e, 0x92 | EFI\_IP6\_PROTOCOL.Transmit() - Transmit() returns EFI\_SUCCESS with valid parameters. | 5.25.5.8.22 to 5.25.5.8.28 belong to one case.  1. Call Transmit()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.8.23 | 0x8f8f115e, 0xd436, 0x41a1, 0xaa, 0x42, 0x11, 0xe7, 0x04, 0xe0, 0x29, 0x11 | Token->Event should be signaled. | 2. Token->Event should be signaled. |
| 5.25.5.8.24 | 0x612b38d1, 0x37cb, 0x419d, 0x8d, 0xfe, 0x44, 0xc7, 0x35, 0xef, 0xe0, 0x17 | Token->Status should be EFI\_SUCCESS. | 3. Token->Status should be EFI\_SUCCESS. |
| 5.25.5.8.25 | 0x464f35de, 0xd546, 0x4140, 0xa7, 0x5e, 0x23, 0xfd, 0xa1, 0xce, 0x2a, 0xd5 | The packet should be received by the other side. | 4. The packet should be received by the other side. |
| 5.25.5.8.26 | 0x0c8799bb, 0xeb02, 0x4172, 0x97, 0xe5, 0xec, 0x6b, 0xaf, 0xe6, 0xe5, 0xa6 | The first fragment of received packet content should be reasonable. | 5. The first fragment of received packet content should be reasonable. |
| 5.25.5.8.27 | 0xe3ececa3, 0x8f49, 0x4bb9, 0xb0, 0xc9, 0x55, 0x85, 0x00, 0x28, 0xc3, 0x1a | The second fragment of received packet content should be reasonable. | 6. The second fragment of received packet content should be reasonable. |
| 5.25.5.8.28 | 0xcf73acd9, 0x0893, 0x4b22, 0x88, 0xcf, 0x42, 0x98, 0x22, 0x0e, 0xc0, 0x6c | Total length should be the sum of two fragment length. | 7. Total length should be the sum of two fragment length. |

### Receive()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.9.1 | 0xa1ca863c, 0x8c68, 0x4afc, 0x8a, 0x97, 0xff, 0x60, 0x3e, 0xef, 0xb4, 0xc9 | EFI\_IP6\_PROTOCOL.Receive() - Receive() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Receive()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.9.2 | 0xa9231505, 0xf3ec, 0x462e, 0xb7, 0x0b, 0x14, 0xb2, 0xc6, 0xa2, 0x23, 0xd8 | EFI\_IP6\_PROTOCOL.Receive() - Receive() returns EFI\_INVALID\_PARAMETER with a NULL Token. | Call Receive()with a NULL Token, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.9.3 | 0xdf7d75d2, 0x4288, 0x4a50, 0xa5, 0xdf, 0x01, 0x85, 0x98, 0x74, 0xb8, 0x29 | EFI\_IP6\_PROTOCOL.Receive() - Receive() returns EFI\_INVALID\_PARAMETER with a NULL Token->Event. | Call Receive()with a NULL Token->Event, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.5.9.4 | 0x1bbc8695, 0x6552, 0x422d, 0xb1, 0x32, 0xda, 0x58, 0x03, 0x0e, 0xf5, 0xb6 | EFI\_IP6\_PROTOCOL.Receive() – Receive() returns EFI\_ACCESS\_DENIED with a Token->Event which has already been in the receive queue. | Call Receive()with a Token->Event which has already been in the receive queue, the return status should be EFI\_ACCESS\_DENIED. |
| 5.25.5.9.5 | 0x5b0a58f2, 0x6668, 0x4247, 0xae, 0x25, 0xae, 0x7e, 0x24, 0x75, 0x02, 0xd7 | EFI\_IP6\_PROTOCOL.Receive() – Receive() returns EFI\_SUCCESS with valid parameters. | 5.25.5.9.5 to 5.25.5.9.11 belong to one case.  1. Call Receive()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.9.6 | 0x019b2b66, 0xfbce, 0x4cab, 0xab, 0x09, 0xd8, 0xdd, 0x34, 0x70, 0x4e, 0xe9 | Token->Event should be signaled. | 2. Token->Event should be signaled. |
| 5.25.5.9.7 | 0x5750bf3b, 0xcead, 0x49a9, 0xad, 0x33, 0xb4, 0x6e, 0x85, 0xc9, 0x78, 0xea | Token->Status should be EFI\_SUCCESS. | 3. Token->Status should be EFI\_SUCCESS. |
| 5.25.5.9.8 | 0x155874a6, 0x0dc9, 0x4b67, 0x9d, 0xb7, 0xda, 0xc9, 0x24, 0xad, 0xc4, 0x4a | Check IPv6 Headlength. | 4. *T*he value of IPv6Headlengthshould be 40*.* |
| 5.25.5.9.9 | 0x7f6044dc, 0x1767, 0x48fc, 0x8a, 0x24, 0xa5, 0x85, 0x6e, 0x82, 0x8e, 0x94 | Check IPv6 RxData.Datalength. | 5. RxData.Datalength should be the same as we expected. |
| 5.25.5.9.10 | 0x022b38cd, 0x5928, 0x4c36, 0x98, 0xd4, 0xd3, 0x67, 0xef, 0x04, 0x55, 0xc7 | RxData.FragmentCount should be 1. | 6. RxData.FragmentCount should be 1. |
| 5.25.5.9.11 | 0x4b71edc9, 0x9c61, 0x45b2, 0xa5, 0x02, 0x05, 0x3a, 0x97, 0x71, 0x19, 0xf3 | The content of Ipv6 header should be the same as we expected. | 7. The content of Ipv6 header should be the same as we expected. |
| 5.25.5.9.12 | 0x48cbff74, 0x89a1, 0x4021, 0xa5, 0x81, 0x40, 0xc1, 0x56, 0xc7, 0x2f, 0x36 | EFI\_IP6\_PROTOCOL.Receive() – Receive() returns EFI\_SUCCESS with valid parameters. | 5.25.5.9.12 to 5.25.5.9.18 belong to one case  1. Call Receive()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.9.13 | 0xa433bb6d, 0x152c, 0x4de8, 0xa6, 0x01, 0x95, 0x31, 0x4d, 0xc3, 0x08, 0xd1 | Token->Event should be signaled. | 2*.* Token->Event should be signaled. |
| 5.25.5.9.14 | 0x0011751a, 0x87f4, 0x4572, 0xad, 0x75, 0xa5, 0x13, 0x84, 0xbf, 0x01, 0x0a | Token->Status should be EFI\_SUCCESS. | *3.* Token->Status should be EFI\_SUCCESS. |
| 5.25.5.9.15 | 0xa2d00870, 0xe59f, 0x4b55, 0xbe, 0x36, 0xda, 0x81, 0x15, 0xe4, 0x57, 0x41 | Check IPv6 Headlength. | 4. The value of IPv6 Headlength should be 40. |
| 5.25.5.9.16 | 0x99aef759, 0xcd2e, 0x46bd, 0x8d, 0x8a, 0x6c, 0xe7, 0x90, 0x8a, 0xf9, 0xa0 | Check IPv6 RxData.Datalength. | 5. RxData.Datalength should be thesame as we expected. |
| 5.25.5.9.17 | 0x1f01211f, 0x1c55, 0x4ee8, 0xb5, 0xe7, 0x14, 0x72, 0xcd, 0xf7, 0x60, 0x64 | RxData.FragmentCount should be 2. | *6.* RxData.FragmentCount should be 1. |
| 5.25.5.9.18 | 0x72f6a9fd, 0xb4bf, 0x47f2, 0x85, 0x07, 0x37, 0x99, 0x02, 0x2f, 0x06, 0xea | The content of Ipv6 header should be the same as we expected. | 7. The content of Ipv6 header should be the same as we expected. |

### Cancel()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.10.1 | 0x136f34b0, 0x4806, 0x4150, 0x98, 0x3c, 0x0c, 0x54, 0x1d, 0x7e, 0x8e, 0x2f | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Cancel()with a Receive Token and a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.10.2 | 0x9c7cacd0, 0xcb07, 0x4181, 0x93, 0x80, 0x90, 0x12, 0xbb, 0x60, 0xe6, 0xe3 | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Cancel()with a Transmit Token and a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.10.3 | 0x5e2ebb02, 0xe419, 0x4ed4, 0xa7, 0xd3, 0xa3, 0xa7, 0xba, 0xb4, 0xee, 0x46 | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_FOUND with a Token which hasn’t been inserted into receive queue. | Call Cancel()with a Token which hasn’t been inserted into receive queue, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.10.4 | 0x7ceb17ac, 0x03bf, 0x427e, 0xbe, 0xe6, 0x98, 0x7f, 0xda, 0x4f, 0x5c, 0x36 | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_FOUND with a Token which hasn’t been inserted into transmit queue. | Call Cancel()with a Token which hasn’t been inserted into transmit queue, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.10.5 | 0x02c484a9, 0x86aa, 0x4484, 0x91, 0xa5, 0x50, 0x0f, 0xd7, 0x0c, 0x3c, 0x84 | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_FOUND with a Token which has been removed from receive queue. | Call Cancel()with a Token which has been removed from receive queue, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.10.6 | 0xf1955578, 0x07ba, 0x4119, 0xbe, 0xa2, 0xe0, 0xb1, 0x2b, 0x41, 0x77, 0x59 | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_FOUND with a Token which has been removed from transmit queue. | Call Cancel()with a Token which has been removed from transmit queue, the return status should be EFI\_NOT\_FOUND. |
| 5.25.5.10.7 | 0xdb1f8413, 0x7d91, 0x4366, 0x94, 0xe7, 0x96, 0xec, 0xf9, 0xd6, 0x0e, 0xbb | EFI\_IP6\_PROTOCOL.Receive() – Receive() returns EFI\_SUCCESS with valid parameters. | 5.25.5.10.7 to 5.25.5.10.10 belong to one case.  1. Call Receive()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.10.8 | 0xb5c49851, 0x0ea9, 0x4d1c, 0x9a, 0xbd, 0x98, 0x5f, 0x94, 0x98, 0x32, 0xf1 | EFI\_IP6\_PROTOCOL.Cancel() – Cancel() returns EFI\_SUCCESS with valid parameters. | 2. Call Cancel()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.25.5.10.9 | 0xff8a1c8f, 0xdf30, 0x4e95, 0xbf, 0x98, 0x11, 0x46, 0xc0, 0xa3, 0xec, 0x50 | Token->Status should be EFI\_ABORTED. | Token->Status should be EFI\_ABORTED. |
| 5.25.5.10.10 | 0x53bb7192, 0xe93a, 0x4a4b, 0xba, 0x2f, 0x58, 0x26, 0x6c, 0xe9, 0xdc, 0x80 | Token->Event should be signaled. | Token->Event should be signaled. |

### Poll()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.5.11.1 | 0xf0a862e2, 0xf222, 0x4742, 0x9e, 0x3f, 0x26, 0xa9, 0x18, 0xd6, 0x9e, 0xf1 | EFI\_IP6\_PROTOCOL.Poll() – Poll() returns EFI\_NOT\_STARTED with a not configured ChildHandle. | Call Poll()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.25.5.11.2 | 0x6ee2f2aa, 0x0a9f, 0x4690, 0xa5, 0x42, 0x95, 0x02, 0x1e, 0x5e, 0xd8, 0xbf | EFI\_IP6\_PROTOCOL.Poll() – Poll() returns EFI\_NOT\_READY with no income and outcome packets. | Call Poll()with no income and outcome packets, the return status should be EFI\_NOT\_READY. |

## EFI\_IP6CONFIG\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_IP6\_CONFIG\_PROTOCOL Section.

### SetData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.6.1.1 | 0x7a224cce,  0xb79b,  0x472a,0x9b,  0x8c,0xa4,  0x7e,0x07,  0x4d,0x5e,  0xef | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI\_INVALID\_PARAMETER with Data beingNULL | Call SetData() with **Data** is **NULL**, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.2 | 0x46f12872,  0x61f2,  0x46e4,0xa2,  0xf9,0x5f,  0x68,0x5b,  0x41,0x94,  0x79 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_INVALID\_PARAMETER with ManualAddress being **::.** | 5.25.6.1.2 to 5.25.6.1.7 belong to one case.  1. Call SetData() with valid parameters except invalid ManualAddress(::), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.3 | 0x1cac93d3,  0x732a,  0x4e30,0x89,  0x4d,0xee,  0x63,0xb6,  0xf4,0x86,  0xa0 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_INVALID\_PARAMETER with ManualAddress containing duplicated entries. | 2. Call SetData() with valid parameters except invalid ManualAddress (2002::5000,2002::5001,2002::5000), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.4 | 0xd005ebf3,  0xcfd6,  0x498a,  0x90,0x05,  0xc2,0xb3,  0x70,0x2e,  0xb4,0xfc | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_INVALID\_PARAMETER with **Gateway** beingmulticast. | 3. Call SetData() with valid parameters except invalid **Gateway** (ff02::1), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.5 | 0x389806d5,  0x4506,  0x4319,  0x8d,0x17,  0x9b,0x4f,  0xc9,0xd9,  0x7e,0x25 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_INVALID\_PARAMETER with **Gateway** containing duplicated entries. | 4. Call SetData() with valid parameters except invalid Gateway (2002::5000,2002::5001,2002::5000), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.6 | 0x5aefdb0c,  0x322f,  0x49c3,  0x9d,0xd2,  0xdf,0xe2,  0x1b,0x66,  0xb3,0x08 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_INVALID\_PARAMETER with **DnsServer** being multicast. | 5. Call SetData() with valid parameters except invalid **DnsServer** (ff02::1), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.7 | 0xd339988f,  0x2595,  0x4fb5,  0x81,0xae,  0xa9,0x4d,  0xc4,0x70,  0xb2,0x34 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_INVALID\_PARAMETER with **DnsServer** containing duplicated entries. | 6. Call SetData() with valid parameters except invalid DnsServer (2002::5000,2002::5001,2002::5000), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.1.8 | 0x4319a43b,  0x7641,  0x47c0,  0x84,0xbb,  0x98,0x5c,  0x47,0x99,  0x02,0xa2 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_WRITE\_PROTECTED when trying to set **InterfaceInfo.** | Call SetData() to set **InterfaceInfo**, The return status should be EFI\_WRITE\_PROTECTED. |
| 5.25.6.1.9 | 0x01f3b344,  0xeb52,  0x4086,  0xb9,0x49,  0x55,0xd7,  0xe4,0xdc,  0x5b,0xde | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_WRITE\_PROTECTED when trying to set **ManualAddress** under **Automatic** policy. | 5.25.6.1.9 to 5.25.6.1.11 belong to one case.  1. Call SetData() to set **MaualAddress(2002::5000)** under **Automatic** policy, The return status should be EFI\_WRITE\_PROTECTED. |
| 5.25.6.1.10 | 0xf612af26,  0x2519,  0x497c,  0xb2,0x05,  0x37,0xa2,  0x91,0x4a,  0xee,0x05 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_WRITE\_PROTECTED when trying to set **Gateway** under **Automatic** policy. | 2. Call SetData() to set **Gateway(2002::5001)** under **Automatic** policy, The return status should be EFI\_WRITE\_PROTECTED. |
| 5.25.6.1.11 | 0x592c1f3d,  0x249e,  0x4654,  0xb4,0xb1,  0x60,0x04,  0x21,0x62,  0x4d,0xd1 | EFI\_IP6CONFIG PROTOCOL. SetData() - SetData()returns EFI\_WRITE\_PROTECTED when trying to set **DnsServer** under **Automatic** policy. | 3. Call SetData() to set **DnsServer(2002::5001)** under **Automatic** policy, The return status should be EFI\_WRITE\_PROTECTED. |
| 5.25.6.1.12 | 0xd70bce29,  0x8026,  0x4e1b,  0xba,0x8b,  0x36,0xa3,  0x13,0xb4,  0x58,0x59 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_BAD BUFFER SIZE when trying to set **ManualAddress** with wrong **DataSize**. | 5.25.6.1.12 to 5.25.6.1.17 belong to one case.  1. Call SetData() to set **ManualAddress(2002::5000)** with **DataSize** being **16**, The return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.6.1.13 | 0xfe793490,  0x53f8,  0x4991,  0x83,0x48,  0xe6,0x24,  0x53,0x0e,  0x83,0xe9 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_BAD BUFFER SIZE when trying to set **Gateway** with wrong **DataSize**. | 2. Call SetData() to set **Gateway(2002::5001)** with **DataSize** being **8**, The return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.6.1.14 | 0x42ccb2ef,  0xd706,  0x4d1a,  0xb2,0x47,  0xf4,0x2b,  0xba,0x99,  0xf7,0x07 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_BAD BUFFER SIZE when trying to set **DnsServer** with wrong **DataSize**. | 3. Call SetData() to set **Gateway(2002::5002)** with **DataSize** being **8**, The return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.6.1.15 | 0x9168cb20,  0xc891,  0x42da,  0xbb,0x9f,  0x7a,0xdb,  0xe4,0x88,  0xb0,0x12 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_BAD BUFFER SIZE when trying to set **AltInterfaceId** with wrong **DataSize**. | 4. Call SetData() to set **AltInterfaceId** with **DataSize** being **1**, The return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.6.1.16 | 0xad058d87,  0x1015,  0x4b2d,  0xa3,0x51,  0x5b,0xd4,  0xb0,0x93,  0x0b,0x7b | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_BAD BUFFER SIZE when trying to set **DadXmits** with wrong **DataSize**. | 5. Call SetData() to set **DadXmits(3)** with **DataSize** being **1**, The return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.6.1.17 | 0x388be3f6,  0xd63e,  0x4cbf,  0xa3,0xd9,  0x3d,0x94,  0x18,0x23,  0x25,0x9b | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_BAD BUFFER SIZE when trying to set **Policy** with wrong **DataSize**. | 6. Call SetData() to set **Policy(Manual)** with **DataSize** being **1**, The return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.25.6.1.18 | 0x2886bae1,  0x383a,  0x400f,  0x8f,0x88,  0x66,0x37,  0x6b,0x2a,  0x0f,0xf5 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_UNSUPPORTED when trying to set **Maximum** | Call SetData() to set **Maximum**, The return status should be EFI\_UNSUPPORTED. |
| 5.25.6.1.19 | 0xd2c61f06,  0x8822,  0x4a09,  0x89,0xa1,  0x7f,0x06,  0x67,0xfc,  0xaf,0x0e | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_ACCESS DENIED when trying to set valid **ManualAddress** with last asynchronous setting not finished. | Intiate asynchronous **ManualAddress** setting process with **DadXmits 20**.  Before the former setting finishes,  Call SetData() to set valid **ManualAddress**, The return status should be EFI\_ACCESS\_DENIED. |
| 5.25.6.1.20 | 0x0a5902da,  0x4142,  0x4494,  0xac,0x66,  0x2b,0x73,  0x1f,0xfe,  0xa6,0x71 | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS when trying to set valid **InterfaceId.** | 5.25.6.1.20 to 5.25.6.1.23 belong to one case.  1. Call SetData() to set **InterfaceId(0:1:2:3:4:5:6:7)**, The return status should be EFI\_SUCCESS. |
| 5.25.6.1.21 | 0xd9a9ef5e,  0xd819,  0x49d0,  0xbb,0x12,  0x25,0xad,  0xec,0x52,  0xdd,0xb3 | Check the set InterfaceId to be as desired | 2. Call GetData() to retrieve **InterfaceId** and validate it to be **(0:1:2:3:4:5:6:7)**, The compare result should be equal. |
| 5.25.6.1.22 | 0x14e96019,  0x0815,  0x4486,  0x91,0x6c,  0xe4,0x40,  0xe1,0x66,  0x62,0x8e | EFI\_IP6CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS when trying to set valid **DadXmits.** | 3. Call SetData() to set **DadXmits(3)**, The return status should be EFI\_SUCCESS. |
| 5.25.6.1.23 | 0x3458bbe0,  0x0d7e,  0x48ec,  0xb3,0x80,  0x2a,0x88,  0x5f,0x44,  0xe1,0x04 | Check the set DadXmits to be as desired | 4. Call GetData() to retrieve **DadXmits** and validate it to be **3**, The compare result should be equal. |

### GetData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.6.2.1 | 0xd15e421d,  0x6228,  0x4fea,  0x8d,0x5a,  0x33,0x0f,  0xff,0x3f,  0x80,0xd2 | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with DataSizebeingNULL | Call GetData() with **DataSize** is **NULL**, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.2.2 | 0x38b36c04,  0x12e9,  0x4e96,  0xb2,0x4f,  0xc4,0x53,  0x85,0x1e,  0x6c,0x1d | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with **Data NULL** and DataSize not zero | Call GetData() with **Data NULL** and **DataSize** is **not zero**, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.2.3 | 0xd05a6c59,  0x617f,  0x4549,  0x96,0x59,  0x4e,0x0c,  0xfc,0x3c,  0x33,0x36 | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_BUFFER TOO SMALL with DataSizesmaller than Data’s actual size. | 5.25.6.2.3 to 5.25.6.2.4 belong to one case  1. Call GetData() to get ManualAddress with **DataSize** is **16**, The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.6.2.4 | 0xed45c2fe,  0x9ec1,  0x4553,  0xaf,0xa4,  0x77,0x1e,  0x9d,0x4f,  0x76,0x11 | The **DataSize** returned by **GetData()** should be equal to the actual size of the specific data type | 2. Check the **DataSize** returned by **GetData()**, it should be equal to (**sizeof EFI\_IP6\_CONFIG\_MANUAL\_ADDRESS**). |
| 5.25.6.2.5 | 0x59118c46,  0x2f2a,  0x4029,  0xab,0xd6,  0x76,0x74,  0x18,0x92,  0x03,0x69 | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_NOT FOUND when the data type doesn’t exist. | Call GetData() to get Maximum, The return status should be EFI\_NOT\_FOUND. |
| 5.25.6.2.6 | 0x55955d09,  0xc806,  0x4777,  0x9f,0xf0,  0x95,0xc0,  0x0e,0x79,  0xac,0x28 | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_NOT READY when trying to get valid **ManualAddress** with last asynchronous setting not finished. | Intiate asynchronous **ManualAddress** setting process with **DadXmits 20**.  Before the former setting finishes,  Call GetData() to get valid **ManualAddress**, The return status should be EFI\_NOT\_READY. |
| 5.25.6.2.7 | 0xfeaac1a0,  0x95bd,  0x4dcb,  0x91,0xc3,  0x9f,0x08,  0x50,0x4b,  0xef,0xa1 | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_SUCCESS when trying to get valid **InterfaceId.** | 5.25.6.2.7 to 5.25.6.2.10 belong to one case.  1. Call SetData() to set **InterfaceId(0:1:2:3:4:5:6:7)**  2. Call GetData() to get **InterfaceId**,The return status should be EFI\_SUCCESS. |
| 5.25.6.2.8 | 0x3649d729,  0xd6d0,  0x456e,  0x84,0xae,  0xc7,0xe7,  0xb8,0x46,  0x43,0x43 | Check the set InterfaceId to be as desired | 3. Validate the retrieved **InterfaceId** to be **(0:1:2:3:4:5:6:7)**, The compare result should be equal. |
| 5.25.6.2.9 | 0x165e79b4,  0xc987,  0x4100,  0x8a,0xa2,  0x8a,0xb1,  0x15,0xb0,  0x7f,0xad | EFI\_IP6CONFIG PROTOCOL.GetData() - GetData()returns EFI\_SUCCESS when trying to get valid **DadXmits.** | 4. Call SetData() to set **DadXmits(3).**  5. Call GetData() to get **DadXmits**. The return status should be EFI\_SUCCESS. |
| 5.25.6.2.10 | 0xdb420311,  0x17f7,  0x40cf,  0xa0,0xb1,  0x02,0x94,  0xd5,0xdc,  0xcc,0x92 | Check the set DadXmits to be as desired | 6. Validate the retrieved **DadXmits** to be **3**, The compare result should be equal. |

### RegisterDataNotify()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.6.3.1 | 0x7e3f6157,  0xec75,  0x4ecd,  0xa7,0x9b,  0x49,0x26,  0xf3,0xaa,  0x1c,0x0d | EFI\_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify()returns EFI\_INVALID\_PARAMETER with **Event** being **NULL** | Call RegisterDataNotify() with **Event** is **NULL**, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.3.2 | 0x70dc8c71,  0xc54d,  0x4446,  0x8a,0xd9,  0xba,0xc0,  0x86,0xe4,  0x3d,0x17 | EFI\_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify()returns EFI\_UNSUPPORTED with **Datatype** not supported | Call RegisterDataNotify() with **Datatype** being **Maximum,** The return status should be EFI\_UNSUPPORTED. |
| 5.25.6.3.3 | 0x2d88f18b,  0x0bef,  0x4616,  0xbd,0xe5,  0xca,0x4e,  0x00,0x86,  0xe1,0xd3 | EFI\_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify()returns EFI\_ACCESS\_DENIED with **Event** already be registered on the same DataType. | 1. Call RegisterDataNotify() with **Datatype** being **Policy** successfully.  2. Call RegisterDataNotify() with **Datatype** being **Policy** and the same **Event** again**,** The return status should be EFI\_ACCESS\_DENIED. |
| 5.25.6.3.4 | 0x9a98dc85,  0xd018,  0x45aa,  0xb8,0x51,  0x34,0xee,  0x2f,0x67,  0x16,0xd4 | EFI\_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify()returns EFI\_SUCCESS with valid parameters | 5.25.6.3.4 to 5.25.6.3.5 belong to one case  1. Call RegisterDataNotify() with **Datatype** being **ManualAddress** successfully. |
| 5.25.6.3.5 | 0x39f7fb37,  0x9f9f,  0x485e,  0x8d,0xbc,  0x0f,0x31,  0x91,0xda,  0x99,0x09 | After the data is set, the **Event** should be signaled correctly. | 2. The **Event** should be signaled and the context of the **Event** should be changed. |
| 5.25.6.3.6 | 0xa13da599,  0x37e7,  0x474a,  0x93,0x43,  0x83,0xc9,  0xef,0xe8,  0x08,0x93 | EFI\_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify()returns EFI\_SUCCESS with valid parameters | 5.25.6.3.6 to 5.25.6.3.9 belong to one case.  1. Call RegisterDataNotify() with **Datatype** being **Policy** successfully. |
| 5.25.6.3.7 | 0x5428bdd5,  0x4332,  0x4e3b,  0x84,0x1f,  0x3e,0x60,  0x54,0x0a,  0xa3,0x5d | EFI\_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify()returns EFI\_SUCCESS with the same **Event**. | 2. Call RegisterDataNotify() with **Datatype** being **DadXmits** and the same **Event** successfully. |
| 5.25.6.3.8 | 0x1844a7c8,  0x730c,  0x4927,  0x8e,0x02,  0xce,0x0a,  0x6c,0xa0,  0x8d,0xcc | After the data is set, the **Event** should be signaled correctly. | 3. Call SetData() to set **Policy**. The **Event** should be signaled and the context should be changed. |
| 5.25.6.3.9 | 0xb0e66591,  0x9076,  0x48e3,  0x8d,0xf6,  0x2a,0x1d,  0x59,0xa5,  0x72,0xdb | After the data is set, the **Event** should be signaled correctly. | 4. Call SetData() to set DadXmits. The **Event** should be signaled and the context should be changed. |

### UnregisterDataNotify()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.6.4.1 | 0x8ab0e5a2,  0xa4e1,  0x4282,  0x87,0xb5,  0xe3,0x77,  0xc7,0x63,  0xad,0x2f | EFI\_IP6CONFIG PROTOCOL.UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_INVALID\_PARAMETER with **Event** being **NULL** | Call UnregisterDataNotify() with **Event** is **NULL**, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.6.4.2 | 0x5c68228f,  0xaaae,  0x4d0b,  0x99,0x27,  0x76,0x64,  0x47,0x6e,  0xf3,0x60 | EFI\_IP6CONFIG PROTOCOL.UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_NOT FOUND with no **Event** registered for the **Datatype**. | Call UnregisterDataNotify() with **Datatype** being **ManualAddress** and the **Event** not registered for the **Datatype** before**,** The return status should be EFI\_NOT\_FOUND. |
| 5.25.6.4.3 | 0x55d8193e,  0xf58e,  0x4800,  0x92,0x4b,  0x73,0xc9,  0x02,0x09,  0x8d,0xd8 | EFI\_IP6CONFIG PROTOCOL.UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_NOT FOUND with **Event** first registered and then unregistered for the Datatype. | 1. Call RegisterDataNotify() with **Datatype** being **ManualAddress** successfully.  2. Call UnregisterDataNotify() with **Datatype** being **ManualAddress** successfully.  3. Call UnregisterDataNotify() with **Datatype** being **ManualAddress** and the same **Event** again**,** The return status should be EFI\_NOT\_FOUND. |
| 5.25.6.4.4 | 0x42eb4628,  0x8df6,  0x4704,  0x81,0xe5,  0xf7,0xea,  0xe6,0xcb,  0xb2,0x70 | EFI\_IP6CONFIG PROTOCOL.UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_SUCCESS with valid parameters. | 1. Call RegisterDataNotify() with **Datatype** being **Policy** successfully.  2. Call UnregisterDataNotify() with **Datatype** being **Policy** successfully. |
| 5.25.6.4.5 | 0x174cec07,  0xe573,  0x434b,  0x8e,0x99,  0x77,0xf8,  0xae,0x9c,  0x55,0xb5 | EFI\_IP6CONFIG PROTOCOL.UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_SUCCESS with valid parameters. | 5.25.6.4.5 to 5.25.6.4.7 belong to one case.  1.Call RegisterDataNotify() with **Datatype** being **Policy** successfully.  2.Call RegisterDataNotify() with **Datatype** being **DadXmits** successfully.  3.Call UnregisterDataNotify() with **Datatype** being **Policy** successfully. |
| 5.25.6.4.6 | 0x1f5ef1af,  0x8a19,  0x48d6,  0x83,0x1f,  0x51,0xbe,  0x00,0xb3,  0x2a,0xa5 | After the data is set, the unregistered **Event** should not be signaled correctly. | 4. Call SetData() to set **Policy**. The **Event** should not be signaled and the context should not be changed. |
| 5.25.6.4.7 | 0x388c8838,  0x7790,  0x4a1f,  0x9d,0xb7,  0x50,0x17,  0xd7,0xaa,  0x60,0xdb | After the data is set, the registered **Event** should be signaled correctly. | 5. Call SetData() to set **DadXmits**. The **Event** should be signaled and the context should be changed. |

## EFI\_IPSEC\_CONFIG\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_IPSEC\_CONFIG\_PROTOCOL Section.

### SetData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.7.1.1 | 0x235a63c3, 0x2ba4, 0x4d1d, 0x8e, 0x25, 0xc8, 0x7e, 0x47, 0x35, 0x36, 0x1c | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData() returns EFI\_UNSUPPORTED with an invalid **DataType** (>2) | Call SetData()with an invalid DataType (>2), The return status should be EFI\_UNSUPPORTED. |
| 5.25.7.1.2 | 0x77f0b145, 0x48a3, 0x4780, 0x8c, 0x0e, 0x63, 0x5b, 0x91, 0x6f, 0x4d, 0xf5 | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS with valid DataType(0)/Selector/Data. | 5.25.7.1.2 to 5.25.7.1.4 belong to one case.  1. Call SetData() with valid DataType(0)/Selector/Data. The return status should be EFI\_SUCCESS. |
| 5.25.7.1.3 | 0x8739610b, 0xabf3, 0x4994, 0x96, 0xee, 0x87, 0xd4, 0x95, 0x27, 0x45, 0x67 | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - returns EFI\_SUCCESS withvalid DataType(0) /Selector and NULL Data. | 2. Call SetData() with validDataType(0) /Selector and NULL Data, The return status should be EFI\_SUCCESS. |
| 5.25.7.1.4 | 0xeb931bcf, 0x074a, 0x4e69, 0x83, 0xee, 0xd3, 0xc6, 0x39, 0xc6, 0x84, 0xef | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - After flush given selector configuration by SetData, GetData()returns EFI\_NOT\_FOUND with valid **DataType**(0) /**Selector/DataSize**. | 3. Call GetData()with valid **DataType**(0) /**Selector/DataSize**, The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.1.5 | 0x35ec56a7, 0x1c1a, 0x4c84, 0xb0, 0x68, 0x40, 0x53, 0x7c, 0x45, 0x95, 0x41 | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS with valid DataType(1)/Selector/Data. | 5.25.7.1.5 to 5.25.7.1.7 belong to one case.  1. Call SetData() with valid DataType(1)/Selector/Data. The return status should be EFI\_SUCCESS. |
| 5.25.7.1.6 | 0x8b6ddfbf, 0x8de1, 0x418d, 0xb0, 0x76, 0xf4, 0x48, 0x07, 0x46, 0xb6, 0x3a | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - SetData()returns EFI\_SUCCESSwithvalidDataType(1) /Selector and NULL Data. | 2. Call SetData() with validDataType(1) /Selector and NULL Data, The return status should be EFI\_SUCCESS. |
| 5.25.7.1.7 | 0xa510e599, 0x2cdd, 0x4c14, 0xbe, 0xc9, 0xbd, 0x2f, 0xd8, 0x7d, 0x50, 0x60 | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - After flush given selector configuration by SetData, GetData()returns EFI\_NOT\_FOUND with valid **DataType**(1) /**Selector**/**DataSize**. | 3. Call GetData()with valid **DataType**(1) /**Selector**/**DataSize**, The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.1.8 | 0x69d0edc5, 0xd259, 0x42ea, 0xa6, 0x97, 0x47, 0x8c, 0x2a, 0x32, 0x0c, 0x08 | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS with valid DataType(2)/Selector/Data. | 5.25.7.1.8 to 5.25.7.1.10 belong to one case.  1. Call SetData() with valid DataType(2)/Selector/Data. The return status should be EFI\_SUCCESS. |
| 5.25.7.1.9 | 0xe389a40e, 0x4c21, 0x4cf1, 0x88, 0xb3, 0xae, 0x86, 0x9b, 0x0b, 0xc2, 0x35 | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - SetData()returns EFI\_SUCCESS withvalid DataType(2) /Selectorand NULL Data. | 2. Call SetData() with validDataType(2) /Selector andNULL Data, The return status should be EFI\_SUCCESS. |
| 5.25.7.1.10 | 0x4d6b9807, 0x4d26, 0x43aa, 0x8a, 0x53, 0xd1, 0xff, 0xe5, 0x2b, 0xb0, 0xde | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - After flush given selector configuration by SetData, GetData()returns EFI\_NOT\_FOUND with valid **DataType**(2) /**Selector**/**DataSize**. | 3. Call GetData()with valid **DataType**(2) /**Selector**/**DataSize**, The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.1.11 | 0x5747257a, 0xabff, 0x4ac4, 0xa9, 0xb0, 0xfc, 0x82, 0xf7, 0xd0, 0xce, 0xa2 | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS with valid DataType(0)/Selector/Data. | 5.25.7.1.11 to 5.25.7.1.13 belong to one case.  1. Call SetData() with valid DataType(0)/Selector/Data. The return status should be EFI\_SUCCESS. |
| 5.25.7.1.12 | 0x808d03fc, 0x2d68, 0x4c51, 0x90, 0x31, 0x01, 0x32, 0x64, 0xf5, 0xf7, 0x85 | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - SetData()returns EFI\_SUCCESS withvalid DataType(0) /Data andNULL Selector. | 2. Call SetData() with validDataType(0) /Data andNULL Selector, The return status should be EFI\_SUCCESS. |
| 5.25.7.1.13 | 0x2f5d587d, 0x4216, 0x42dd, 0x92, 0x41, 0x72, 0x60, 0xe9, 0x65, 0xa6, 0xf6 | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - After flush entire configuration by SetData, GetData()returns EFI\_NOT\_FOUND with valid **DataType**(0) /**Selector**/**DataSize**. | 3. Call GetData()with valid **DataType**(0) /**Selector**/**DataSize**, The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.1.14 | 0x39a5db14, 0xebb0, 0x460f, 0x92, 0x99, 0x36, 0x28, 0x3f, 0x51, 0x9d, 0xff | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS with valid DataType(1)/Selector/Data. | 5.25.7.1.14 to 5.25.7.1.16 belong to one case.  1. Call SetData() with valid DataType(1)/Selector/Data. The return status should be EFI\_SUCCESS. |
| 5.25.7.1.15 | 0xdee52264, 0x3da1, 0x4f5d, 0xa2, 0x43, 0xa1, 0x15, 0xad, 0xd3, 0x3f, 0x40 | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - SetData()returns EFI\_SUCCESSwithvalid DataType(1) /Data andNULL Selector. | 2. Call SetData() with validDataType(1) /DataandNULL Selector, The return status should be EFI\_SUCCESS. |
| 5.25.7.1.16 | 0xd76b9b01, 0x6649, 0x4b43, 0xa0, 0x05, 0x1a, 0x64, 0x69, 0xc3, 0xef, 0x0f | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - After flush entire configuration by SetData, GetData()returns EFI\_NOT\_FOUND with valid **DataType**(1) /**Selector**/**DataSize**. | 3. Call GetData()with valid **DataType**(1) /**Selector**/**DataSize**, The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.1.17 | 0x5f9e36d3, 0xa945, 0x4b20, 0xa2, 0x9b, 0x30, 0x3e, 0x9b, 0xd5, 0x6c, 0xcd | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - SetData()returns EFI\_SUCCESS with valid DataType(2)/Selector/Data. | 5.25.7.1.17 to 5.25.7.1.19 belong to one case.  1. Call SetData() with valid DataType(2)/Selector/Data. The return status should be EFI\_SUCCESS. |
| 5.25.7.1.18 | 0xaec61686, 0xf303, 0x4697, 0xb0, 0x7d, 0xe2, 0x08, 0x8e, 0x52, 0x05, 0x58 | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - SetData()returns EFI\_SUCCESS withvalid DataType(2) /Data and NULL Selector. | 2. Call SetData() with validDataType(2) /Data and NULL Selector, The return status should be EFI\_SUCCESS. |
| 5.25.7.1.19 | 0x69c4e05f, 0x7b94, 0x4c82, 0x81, 0x47, 0xd9, 0x14, 0x57, 0x86, 0x24, 0x3f | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() - After flush entire configuration by SetData, GetData()returns EFI\_NOT\_FOUND with valid **DataType**(2) /**Selector**/**DataSize**. | 3. Call GetData()with valid **DataType**(2) /**Selector**/**DataSize**, The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.1.20 | 0x486c7a3e, 0x4a65, 0x4da6, 0x8e, 0x52, 0x6b, 0x64, 0x48, 0xc3, 0x68, 0xaa | EFI\_IPSEC\_CONFIG PROTOCOL.SetData()returnsEFI\_SUCCESS validDataType(1)/Selector/SA\_Data2 | 5.25.7.1.20 to 5.25.7.1.22  belong to one case.  1. Call SetData()with valid DataType(1)/Selector/  SA\_Data2. The return status should be EFI\_SUCCESS |
| 5.25.7.1.21 | 0x92302107, 0x20fa, 0x49b9, 0x84, 0x5f, 0xec, 0xc6, 0xe0, 0x28, 0x31, 0xf3 | EFI\_IPSEC\_CONFIG PROTOCOL. SetData () - SetData()returns EFI\_SUCCESSwith valid DataType(1) /Selector and NULL Data. | 2. Call SetData()with valid DataType(1) /Selector  and NULL Data, The return  status should be EFI\_SUCCESS. |
| 5.25.7.1.22 | 0x03b2df9d, 0xe5c1, 0x47b3, 0xaa, 0x7a, 0xa0, 0xbb, 0x1d, 0xf2, 0xf0, 0x9b | EFI\_IPSEC\_CONFIG PROTOCOL.SetData() **-** After flush given selector configuration by SetData**,** GetData()returns EFI\_NOT\_FOUNDwith valid DataType(1) /Selector/DataSize. | 3. Call GetData()with valid DataType(1)/Selector/  DataSize, The return status  should be EFI\_NOT\_FOUND. |

### GetData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.7.2.1 | 0xa8339798, 0x45fa, 0x47a8, 0xaf, 0x9e, 0x74, 0x17, 0xcd, 0x78, 0xef, 0x40 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL Selector. | Call GetData() with NULL Selector, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.2 | 0x1d04e3e9, 0xfc36, 0x4321, 0xa8, 0x22, 0x51, 0xb2, 0x59, 0x01, 0xbf, 0xb0 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_UNSUPPORTED with an invalid **DataType** (>2) | Call SetData()with an invalid DataType(>2), The return status should be EFI\_UNSUPPORTED. |
| 5.25.7.2.3 | 0x4da58bcc, 0x1ae2, 0x450d, 0xbc, 0x1b, 0x0d, 0x76, 0x77, 0x3a, 0xab, 0x79 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL Data. | Call GetData()with NULL Data, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.4 | 0x39962424, 0x200d, 0x40cd, 0x8f, 0x5b, 0xfd, 0x3f, 0xf8, 0xaa, 0x51, 0x96 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL DataSize. | Call GetData()with NULL DataSize, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.5 | 0x1ef8f8fb, 0xf494, 0x4411, 0x87, 0xd2, 0x73, 0x43, 0x88, 0x6a, 0x14, 0xe7 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_BUFFER\_TOO\_SMALL with small DataSize. | Call GetData()with small DataSize, The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.7.2.6 | 0xddc718a3, 0xb10d, 0x4f05, 0x9d, 0x97, 0x65, 0xda, 0x75, 0xd9, 0x02, 0xca | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL Data. | Call GetData()with NULL Data, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.7 | 0xc6d16b39, 0x34f6, 0x438a, 0xa5, 0x77, 0xbf, 0xd3, 0x13, 0xbc, 0x9e, 0xe8 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL DataSize. | Call GetData()with NULL DataSize, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.8 | 0xa5fecb65, 0x0501, 0x4d66, 0xbe, 0x1c, 0x37, 0xac, 0xb7, 0x8a, 0xd4, 0xe8 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_BUFFER\_TOO\_SMALL with small DataSize. | Call GetData()with small DataSize, The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.7.2.9 | 0x6b1c7e3e, 0x47e7, 0x40ef, 0x85, 0xec, 0x3b, 0x8c, 0x0f, 0xa6, 0x08, 0x1f | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL Data. | Call GetData()with NULL Data, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.10 | 0xb4138aae, 0xccfb, 0x45af, 0xa6, 0x41, 0x0a, 0x1c, 0x7f, 0x9d, 0x86, 0x1b | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_INVALID\_PARAMETER with NULL DataSize. | Call GetData()with NULL DataSize, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.2.11 | 0xea851d2d, 0x4031, 0x4966, 0x91, 0x8e, 0x24, 0xda, 0x2a, 0x56, 0xc3, 0xb7 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_BUFFER\_TOO\_SMALL with small DataSize. | Call GetData()with small DataSize, The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.7.2.12 | 0xd2cabfe5,  0x85a0, 0x47a1, 0x8d,0x71, 0x3c,0x3f, 0x64,0x4a, 0x41,0xf3 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() **-** GetData()returns EFI\_INVALID\_PARAMETERwith NULL SA\_DATA2 | Call GetData()with NULL SA\_DATA2, The return status should be EFI\_INVALID\_PARAMETER . |
| 5.25.7.2.13 | 0x91591c0, 0x5a13, 0x448e, 0xbf, 0x21, 0x1d, 0x12, 0xb3, 0x8c, 0x9e, 0x6d | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() **-** GetData()returns EFI\_INVALID\_PARAMETER with NULL SA\_DATA2 datasize | Call GetData()with NULL  SA\_DATA2 datasize, The  return status should be  EFI\_INVALID\_PARAMETER |
| 5.25.7.2.14 | 0x64ec8c85, 0x7661, 0x4364, 0xa1, 0xf3, 0x56, 0x62, 0x69, 0x3d, 0x8a, 0x7a | EFI\_IPSEC\_CONFIG PROTOCOL**.**GetData() **-** GetData()returns EFI\_BUFFER\_TOO\_SMALLwith small SA\_DATA2  datasize | Call GetData()with small  SA\_DATA2 datasize, The  return status should be  EFI\_BUFFER\_TOO\_SMALL . |
| 5.25.7.2.15 | 0x437749ac, 0x27bc, 0x46ac, 0xb7, 0xa1, 0x1b, 0x39, 0xee, 0xcc, 0x58, 0xc0 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_SUCCESS with Valid DataType(0)/Selector/DataSize. | Call GetData()with Valid DataType(0)/Selector/DataSize. The return status should be EFI\_SUCCESS. |
| 5.25.7.2.16 | 0xe53c2379, 0x58fb, 0x402f, 0xbb, 0x47, 0x12, 0xd7, 0xe3, 0x55, 0x8d, 0x01 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns the right values which are set before. | Call GetData()with Valid DataType(0)/Selector/DataSize. The right values should be same as the values which are set before. |
| 5.25.7.2.17 | 0x37f06d59, 0x2e1f, 0x4ccd, 0x83, 0xbc, 0x1b, 0xf2, 0xcf, 0x4b, 0x92, 0x4e | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_SUCCESS with Valid DataType(1)/Selector/DataSize. | Call GetData()with Valid DataType(1)/Selector/DataSize. The return status should be EFI\_SUCCESS. |
| 5.25.7.2.18 | 0x077a8be2, 0xdd60, 0x48b5, 0xaf, 0x2e, 0x05, 0xcd, 0xc7, 0x07, 0x64, 0xf0 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns the right values which are set before. | Call GetData()with Valid DataType(1)/Selector/DataSize. The right values should be same as the values which are set before. |
| 5.25.7.2.19 | 0x35adfec2, 0x5c65, 0x431f, 0x87, 0x86, 0x7b, 0x70, 0x81, 0x69, 0x71, 0xba | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns EFI\_SUCCESS with Valid DataType(2)/Selector/DataSize. | Call GetData()with Valid DataType(2)/Selector/DataSize. The return status should be EFI\_SUCCESS. |
| 5.25.7.2.20 | 0x26a81e68, 0x1aec, 0x4f1f, 0x9c, 0xe5, 0xc1, 0x59, 0xf2, 0xf3, 0xea, 0x12 | EFI\_IPSEC\_CONFIG PROTOCOL.GetData() - GetData()returns the right values which are set before. | Call GetData()with Valid DataType(2)/Selector/DataSize. The right values should be same as the values which are set before. |
| 5.25.7.2.21 | 0x378cd479, 0x2dd4, 0x4bc8,0x9b, 0xd8, 0x8c, 0x23, 0xfd, 0xda, 0x5d, 0x20 | EFI\_IPSEC\_CONFIG\_PROTO COL.GetData- GetData()returns EFI\_SUCCESS with valid DataType(1)/Selector/DataSize | Call GetData() with valid DataType(1)/Selector/DataSize, The return status should be EFI\_SUCCESS. |
| 5.25.7.2.22 | 0x34fc6d63, 0xb2ec, 0x4c20, 0xb7, 0x7d, 0xa8, 0xf8, 0xf, 0x74, 0x7b, 0xa3 | EFI\_IPSEC\_CONFIG\_PROTO COL.GetData- GetData() returns EFI\_SUCCESS& the right SA\_DATA2 which are set before | Call GetData()returns the right SA\_DATA2 which are set before, The return status should be EFI\_SUCCESS**.** |

### GetNextSelector ()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.7.3.1 | 0xf85ce018, 0x2fad, 0x4b4e, 0xbb, 0xbb, 0x1c, 0x59, 0x57, 0x12, 0x85, 0xac | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector()returns EFI\_UNSUPPORTED with an invalid **DataType** (>2) | Call GetNextSelector()with an invalid **DataType** (>2). The return status should be EFI\_UNSUPPORTED. |
| 5.25.7.3.2 | 0x17a12f39, 0xba49, 0x4abb, 0x8f, 0x52, 0x3a, 0x32, 0x24, 0x8e, 0x04, 0xdd | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_INVALID\_PARAMETER with **NULL SelectorSize**. | Call GetNextSelector() with **NULL SelectorSize**. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.3.3 | 0xc404ce41, 0x6802, 0x415d, 0x8b, 0x76, 0x41, 0x26, 0x65, 0x1d, 0x56, 0x29 | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_INVALID\_PARAMETER with **NULL Selector**. | Call GetNextSelector()with **NULL Selector**. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.3.4 | 0x23b72aad, 0xa975, 0x4500, 0x95, 0x19, 0x2e, 0x6d, 0xc4, 0x5f, 0x23, 0x27 | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_BUFFER\_TOO\_SMALL with valid **DataType**(0)/**Selector** and **SelectorSize** is 0. | Call GetNextSelector()with valid **DataType**(0)/**Selector** and **SelectorSize** is 0. The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.7.3.5 | 0xa11a6002, 0x911b, 0x4702, 0x85, 0xa7, 0xc9, 0x73, 0x91, 0xa6, 0xdb, 0x6d | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_BUFFER\_TOO\_SMALL with valid **DataType**(1)/**Selector** and **SelectorSize** is 0. | Call GetNextSelector()with valid **DataType**(1)/**Selector** and **SelectorSize** is 0. The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.7.3.6 | 0xccbcee8b, 0xf23b, 0x4c70, 0x8e, 0x3b, 0x19, 0xdb, 0xa6, 0xd1, 0xa8, 0x51 | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_BUFFER\_TOO\_SMALL with valid **DataType**(2)/**Selector** and **SelectorSize** is 0. | Call GetNextSelector()with valid **DataType**(2)/**Selector** and **SelectorSize** is 0. The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.25.7.3.7 | 0x502ad851, 0x41ae, 0x483e, 0xaa, 0xcd, 0x8d, 0x23, 0x73, 0x04, 0x91, 0xcf | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_SUCCESS with valid **DataType**(0)/**Selector** and **SelectorSize**. | Call GetNextSelector()with valid **DataType**(0)/**Selector** and **SelectorSize**. The return status should be EFI\_SUCCESS. |
| 5.25.7.3.8 | 0x2f0d92f8, 0x2371, 0x4547, 0xa9, 0x5e, 0x79, 0x09, 0xc8, 0x62, 0xee, 0x26 | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_SUCCESS with valid **DataType**(0)/**Selector** and **SelectorSize**. | Call GetNextSelector()with valid **DataType**(0)/**Selector** and **SelectorSize**. The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.3.9 | 0xdaa5a475, 0x0d4a, 0x4e58, 0xa4, 0xd4, 0xfe, 0x33, 0xe7, 0x13, 0xd5, 0xbd | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_SUCCESS with valid **DataType**(1)/**Selector** and **SelectorSize**. | Call GetNextSelector() with valid **DataType**(1)/**Selector** and **SelectorSize**. The return status should be EFI\_SUCCESS. |
| 5.25.7.3.10 | 0x78ea1b63, 0x979e, 0x41fe, 0xab, 0xb1, 0xc3, 0xb3, 0x42, 0x38, 0xc2, 0xa0 | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_SUCCESS with valid **DataType**(1)/**Selector** and **SelectorSize**. | Call GetNextSelector() with valid **DataType**(1)/**Selector** and **SelectorSize**. The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.3.11 | 0xd570e742, 0x8122, 0x4abc, 0xbb, 0xe8, 0x34, 0xcf, 0x8f, 0x6e, 0x00, 0xdd | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_SUCCESS with valid **DataType**(2)/**Selector** and **SelectorSize**. | Call GetNextSelector()with valid **DataType**(2)/**Selector** and **SelectorSize**. The return status should be EFI\_SUCCESS. |
| 5.25.7.3.12 | 0xb3a7efaa, 0x0c6e, 0x4686, 0xad, 0x77, 0xab, 0xd2, 0x62, 0xb4, 0x71, 0xfb | EFI\_IPSEC\_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI\_SUCCESS with valid **DataType**(2)/**Selector** and **SelectorSize**. | Call GetNextSelector()with valid **DataType**(2)/**Selector** and **SelectorSize**. The return status should be EFI\_NOT\_FOUND. |

### RegisterDataNotify ()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.7.4.1 | 0x22857d7f, 0xa20c, 0x467f, 0xa5, 0x70, 0x54, 0xbd, 0x56, 0x3d, 0x93, 0x7e | EFI\_IPSEC\_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify()returns EFI\_INVALID\_PARAMETER with **NULL Event**. | Call RegisterDataNotify()with **NULL Event**. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.4.2 | 0x9361ecca, 0xf59a, 0x4d4c, 0xb5, 0x9d, 0x1a, 0xc8, 0xf3, 0x7b, 0x75, 0x1a | EFI\_IPSEC\_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify()returns EFI\_UNSUPPORTED with invalid **DataType**(>2). | Call RegisterDataNotify() with invalid **DataType**(>2). The return status should be EFI\_UNSUPPORTED. |
| 5.25.7.4.3 | 0x9bd0dce3, 0x15c1, 0x4104, 0x82, 0x3f, 0x35, 0x80, 0x97, 0x00, 0x49, 0xcb | EFI\_IPSEC\_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify()returns EFI\_SUCCESS with valid DataType/Event. | Call RegisterDataNotify()with valid DataType/Event. The return status should be EFI\_SUCCESS. |
| 5.25.7.4.4 | 0x53fe8163, 0xb212, 0x4c7e, 0x88, 0xa0, 0xe9, 0x90, 0x0a, 0x10, 0x20, 0x75 | EFI\_IPSEC\_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify()returns EFI\_ ACCESS\_DENIED with valid DataType/Event. | Call RegisterDataNotify()with valid DataType/Event. The return status should be EFI\_ACCESS\_DENIED. |
| 5.25.7.4.5 | 0xe3ef592d, 0xb247, 0x417f, 0xad, 0x54, 0x4e, 0xfc, 0x0b, 0x7a, 0x03, 0x02 | EFI\_IPSEC\_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify()returns EFI\_SUCCESS with valid DataType/Event. | Call RegisterDataNotify()with valid DataType/Event. The return status should be EFI\_SUCCESS. |

### UnregisterDataNotify ()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.7.5.1 | 0x4fd58448, 0x8d87, 0x4bd0, 0xbf, 0xd1, 0xe0, 0xa5, 0x7a, 0x70, 0xce, 0x0c | EFI\_IPSEC\_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_INVALID\_PARAMETER with **NULL Event**. | Call UnregisterDataNotify()with **NULL Event**. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.25.7.5.2 | 0x12dd249e, 0xa481, 0x4a9a, 0x87, 0x45, 0xa9, 0xfd, 0x26, 0xac, 0xb1, 0xc8 | EFI\_IPSEC\_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_UNSUPPORTED with invalid **DataType**(>2). | Call UnregisterDataNotify()with invalid **DataType**(>2). The return status should be EFI\_UNSUPPORTED. |
| 5.25.7.5.3 | 0xa561620c, 0xfc80, 0x478d, 0xab, 0x8c, 0x2c, 0xdb, 0xc8, 0x47, 0x46, 0xc4 | EFI\_IPSEC\_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_NOT\_FOUND with valid DataType/Event. | Call UnregisterDataNotify()with valid DataType/Event. The return status should be EFI\_NOT\_FOUND. |
| 5.25.7.5.4 | 0x3053b6d9, 0xa5ba, 0x41c1, 0xad, 0x8f, 0x49, 0xf3, 0x37, 0x9f, 0x90, 0x55 | EFI\_IPSEC\_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify()returns EFI\_SUCCESS with valid DataType/Event. | Call UnregisterDataNotify()with valid DataType/Event. The return status should be EFI\_SUCCESS. |
| 5.25.7.5.5 | 0xa829c13e, 0x551d, 0x443e, 0xaf, 0xa0, 0x1d, 0x8d, 0x0a, 0xea, 0x61, 0x98 | EFI\_IPSEC\_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify()returnsEFI\_NOT\_FOUND with valid DataType/Event. | Call UnregisterDataNotify()with valid DataType/Event. The return status should be EFI\_NOT\_FOUND. |

## EFI\_IPSEC2\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_IPSEC2\_PROTOCOL Section.

### ProcessExt()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.25.8.1.1 | 0x5de601fb, 0xc3c4, 0x4bff, 0x89, 0x3e, 0xdd, 0x40, 0x67, 0xd1, 0xe1, 0x6b | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_INVALID\_PARAMETER with NULL OptionsBuffer Input | 1. Call ProcessExt() with NULL OptionsBuffer Input. 2.The return code should be EFI\_INVALID\_PARAMETER |
| 5.25.8.1.2 | 0xd7cf3852, 0xcb7c, 0x4f68, 0x9b, 0x28, 0x56, 0x64, 0x72, 0xbe, 0xe3, 0x3d | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_INVALID\_PARAMETER with NULL OptionsLengthInput | 1. Call ProcessExt() with NULL OptionsLength Input. 2. The return code should be EFI\_INVALID\_PARAMETER |
| 5.25.8.1.3 | 0xf33aeb54, 0xe1be, 0x4541, 0xac, 0x79, 0x4e, 0xc1, 0xbc, 0x23, 0x87, 0x2b | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt()returns EFI\_INVALID\_PARAMETER with NULL FragmentTable Input | 1. Call ProcessExt() with Null FragmentTable Input. 2.The return code should be EFI\_INVALID\_PARAMETER |
| 5.25.8.1.4 | 0x861f3f9, 0x4361, 0x4a23, 0x98, 0x41, 0xf0, 0x2d, 0x14, 0x97, 0x33, 0xb6 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_INVALID\_PARAMETER with NULL **FragmentCount** Input | 1. Call ProcessExt() with NULL FragmentCount Input. 2. The return code should be EFI\_INVALID\_PARAMETER |
| 5.25.8.1.5 | 0x2b45f62a, 0xb9f, 0x473d, 0xbb, 0x5f, 0xcf, 0x59, 0x35, 0xed, 0xae, 0x4a | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode OutBound Call to do IP4 IPSEC with Encrypt Algorithm {SHA1HMAC, 3DESCBC} | 1. Call ProcessExt()in Transport Mode OutBound Call to do IP4 IPSEC with Encrypt Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS |
| 5.25.8.1.6 | 0xd486fd03, 0x7888, 0x42ed, 0x8f, 0xdd, 0xc5, 0xb, 0x40, 0xae, 0x25, 0xd7 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC} and check if Packet Header content is intact | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS and Packet Header content is intact. |
| 5.25.8.1.7 | 0xfd4a5c6f, 0x9072, 0x463a, 0xb6, 0x5, 0x80, 0x72, 0x80, 0x14, 0x13, 0xc9 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt()returns EFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check if Packet Payload Content is intact | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS and Packet Payload Content is intact. |
| 5.25.8.1.8 | 0xbcddcd9a, 0xc0d9, 0x450c, 0xbc, 0xdb, 0xe0, 0xeb, 0x1c, 0xb7, 0x98, 0x3d | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Transport Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC} | 1.Call ProcessExt() in Transport Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}  2.The return code should be EFI\_SUCCESS. |
| 5.25.8.1.9 | 0xd89ad072, 0xfd5e, 0x42af, 0x83, 0x4a, 0xf2, 0xde, 0xcb, 0xfd, 0x9, 0x2d | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Header content is intact | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}.  2.The return code should beEFI\_SUCCESS and Packet Header content is intact |
| 5.25.8.1.10 | 0x530369c, 0xaf77, 0x4064, 0xbc, 0xc1, 0x70, 0x68, 0x31, 0x4, 0x76, 0x94 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Header content is intact | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}.  2.The return code should be EFI\_SUCCESS and Packet Header content is intact |
| 5.25.8.1.11 | 0x6d729b2d, 0x1524, 0x49ae, 0xb6, 0xb9, 0xfa, 0xee, 0x59, 0x51, 0xe1, 0x61 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, 3DESCBC} | 1.Call ProcessExt() in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS. |
| 5.25.8.1.12 | 0x79eba4f0, 0xcfd0, 0x42fa, 0xb7, 0x94, 0x21, 0xa2, 0xd9, 0xac, 0xfa, 0x34 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check Returned Packet Header is set ZERO | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS and Returned Packet Header is set ZERO. |
| 5.25.8.1.13 | 0xd23154b3, 0xbe46, 0x4924, 0x86, 0xfa, 0x1b, 0x16, 0x25, 0x24, 0xfe, 0xc6 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP4 Packet InnerHeader is correct. | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESSand IP4 Packet InnerHeader is correct. |
| 5.25.8.1.14 | 0xf5503af0, 0x8305, 0x40ce, 0x88, 0xf3, 0x29, 0x1a, 0xe, 0x32, 0x5b, 0x9d | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP4 Packet PayLoad is intact. | 1.Call **ProcessExt()** in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI\_SUCCESS and Check IP4 Packet PayLoad is intact. |
| 5.25.8.1.15 | 0x123fa8ee, 0xa9ff, 0x4fa3, 0x92, 0xef, 0x5c, 0x31, 0x60, 0x8c, 0x9e, 0x65 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC} | 1.Call **ProcessExt()** in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC} 2.The return code should be **EFI\_SUCCESS**. |
| 5.25.8.1.16 | 0xbb52fb61, 0xdba9, 0x45b0, 0x9e, 0xb4, 0x2b, 0xfa, 0x1e, 0xa3, 0xa6, 0xde | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check Returned Packet Header is set ZERO | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI\_SUCCESS & Returned Packet Header should be set ZERO. |
| 5.25.8.1.17 | 0x6fc08962, 0xcf2, 0x445b, 0x9f, 0x54, 0x59, 0x12, 0x79, 0xc3, 0xd9, 0x56 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check IP4 Packet InnerHeader is correct | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI\_SUCCESS & IP4 Packet InnerHeader is correct. |
| 5.25.8.1.18 | 0x16dc1d54, 0x755b, 0x482b, 0xa2, 0xca, 0x9d, 0xce, 0xf7, 0xf, 0xa8, 0x8b | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in IPSEC Encrypt & Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC} and Check IP4 Packet PayLoad is intact | 1.Call **ProcessExt()** in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI\_SUCCESS & IP4 Packet PayLoad is intact. |
| 5.25.8.1.19 | 0x5c8f633, 0xea97, 0x4c28, 0xb6, 0xf6, 0x4a, 0xa3, 0x8, 0x7c, 0x9b, 0x52 | **E**FI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Transport Mode OutBound Call to do IPSEC IP6 Packet Encrypt Algorithm {SHA1HMAC, 3DESCBC} | 1.Call ProcessExt() in Transport Mode OutBound Call to do IPSEC IP6 Packet Encrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI\_SUCCESS. |
| 5.25.8.1.20 | 0x25181e14, 0xb84b, 0x4aae, 0x89, 0xdd, 0x4a, 0xe, 0xe0, 0x27, 0xca, 0xc1 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode IPSEC IP6 InBound to Decrypt Algorithm {SHA1HMAC, 3DESCBC}. Check if Packet Header content is intact | 1.Call **ProcessExt()** in Transport Mode IPSEC IP6 InBound to Decrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI\_SUCCESS & Packet Header content is intact. |
| 5.25.8.1.21 | 0xf6ee80b9, 0x622c, 0x4306, 0xae, 0xd2, 0xb6, 0xf8, 0x42, 0x87, 0x92, 0x11 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. Check if Packet Payload Content is intact | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS & Packet Payload Content is intact. |
| 5.25.8.1.22 | 0xf251fd3b, 0xf026, 0x4040, 0x8d, 0x8, 0xc9, 0x22, 0x22, 0xaf, 0xe9, 0xbb | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode OutBound Call to do IPSEC IP6 packet Encrypt Algorithm {SHA1HMAC, AESCBC}. | 1.Call ProcessExt() in Transport Mode OutBound Call to do IPSEC IP6 packet Encrypt Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI\_SUCCESS. |
| 5.25.8.1.23 | 0x5b865ed2, 0x95a6, 0x47bf, 0xbb, 0x35, 0x1a, 0x3b, 0x5, 0x3, 0xb6, 0x80 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returnsEFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Header content is intact. | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}.  2.The return code should be EFI\_SUCCESS& Packet Header content is intact. |
| 5.25.8.1.24 | 0xed35f3c3, 0x2222, 0x4d4c, 0xb1, 0x16, 0x4c, 0x38, 0x25, 0x29, 0x88, 0x4f | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Payload Content is intact. | 1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI\_SUCCESS & Packet Payload Content is intact. |
| 5.25.8.1.25 | 0xb20f0b, 0xdce8, 0x4c22, 0x98, 0x20, 0xcc, 0xb6, 0x5a, 0x40, 0x14, 0xbe | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode OutBound Call to do IP6 IPSEC Tunnel Mode Encrypt Algorithm {SHA1HMAC, 3DESCBC}. | 1.Call ProcessExt() in Tunnel Mode OutBound to do IP6 IPSEC Tunnel Mode Encrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI\_SUCCESS. |
| 5.25.8.1.26 | 0x52ae482f, 0x4882, 0x4945, 0x88, 0xfd, 0x75, 0xe5, 0x8a, 0x14, 0x4a, 0x4f | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP6 Packet InnerHeader is correct. | 1.Call ProcessExt() in Tunnel Mode IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI\_SUCCESS & IP6 Packet InnerHeader is correct. |
| 5.25.8.1.27 | 0xead97223, 0x1dca, 0x4895, 0xa5, 0x9a, 0xc0, 0x3e, 0x8, 0x80, 0x61, 0x54 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP6 Packet PayLoad is intact. | 1.Call ProcessExt() in IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}.  2.The return code should be EFI\_SUCCESS & IP6 Packet PayLoad is intact. |
| 5.25.8.1.28 | 0xd4f53e8f, 0xe53, 0x44ae, 0xbc, 0xef, 0x7e, 0x28, 0xd2, 0x85, 0xc6, 0xf | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt()returns EFI\_SUCCESS in Tunnel Mode OutBound Call to do IP6 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}. | 1.Call ProcessExt() in Tunnel Mode OutBound Call to do IP6 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}.  2.The return code should be EFI\_SUCCESS. |
| 5.25.8.1.29 | 0xd96aaf71, 0xca6f, 0x4cc7, 0x89, 0xf4, 0x99, 0x1a, 0xb1, 0xb5, 0x22, 0xe9 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check Returned Packet Header is set ZERO. | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}.  2.The return code should be EFI\_SUCCESS & Returned Packet Header is set ZERO. |
| 5.25.8.1.30 | 0xc0ca611c, 0x97bb, 0x4c4e, 0x90, 0x84, 0xff, 0x90, 0x94, 0x20, 0xd9, 0x6e | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt()returnsEFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check IP6 Packet InnerHeader is correct. | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}.  2.The return code should be EFI\_SUCCESS & IP6 Packet InnerHeader is correct. |
| 5.25.8.1.31 | 0x6098f2af, 0xe85c, 0x4201, 0xbb, 0xc9, 0xf9, 0x10, 0x2b, 0xcb, 0x94, 0xe7 | EFI\_IPSEC2\_PROTOCOL. ProcessExt–ProcessExt() returns EFI\_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check IP6 Packet PayLoad is intact. | 1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}.  2.The return code should be EFI\_SUCCESS & IP6 Packet PayLoad is intact. |