TR-143 on Verizon FiOS Router (BHR)

ECR

December 2015

Revision History

Version	Contact	Action	Recipient
v1, 02/2013, initial draft	Lawrence Jones		
V2, 10/02015	Lawrence Jones, Richard Jia	Update with high throughput UDP testing	
V3 12/2015	Lawrence Jones	More update after discussion with implementation team	

Table of contents

1. Introduction

This document describes the functional requirements for implementing the monitoring capabilities specified in Broadband Forum specification TR-143, "Enabling Network Throughput Performance Tests and Statistical Monitoring", in the Broadband Home Router (BHR). In particular, the test procedures and variables of interest are documented. TR-143 defines two categories of tests: (1) UDP Echo Plus, and (2) download and upload throughput tests (using FTP or HTTP protocol). While the UDP Echo Plus tests are network initiated (i.e. tests initiated by an external network test system) with the BHR functioning as a responder, the throughput tests are CPE initiated (i.e. test initiated via the ACS environment).

Some additional requirements are included that augment the functionality specified in TR-143. These requirements are included to bypass the dependency on the ACS for those tests that are CPE initiated. In particular, a scheduling mechanism is described to enable BHRs to continually run tests in a recurring fashion and to report test results directly to a test server.

2. TR-143 Proper, Functional Requirements

2.1 General Requirements

[2.1.1] The device must support the download and upload network throughput tests (i.e. .DownloadDiagnostics. and .UploadDiagnostics. objects) and collection of all results variables defined in Table 1 of TR-143 using the HTTP protocol.

[2.1.1.1] The device must be capable of performing a HTTP download test at a speed equal to or exceeding 80% of the port speed (e.g. 80Mbps on a 100Mbps port, 800Mbps on a GigE port, etc …) at 20 msec round trip time. Multithreaded operation (i.e. multiple TCP connections) may be used to achieve this speed 80% port speed requirement if necessary. However, if multiple connections are required to achieve the 80% port speed requirement, it should be noted. Moreover the TCP stack may need to be optimized in some fashion to achieve maximum speed. If the TCP parameters need to be changed from their default values to meet the service tier speed, than these parameter settings should be noted.

[2.1.1.2] The device must be capable of performing a HTTP upload test at a speed equal to or exceeding 80% of the port speed (e.g. 80Mbps on a 100Mbps port, 800Mbps on a GigE port, etc …) at 20 msec round trip time. Multithreaded operation (i.e. multiple TCP connections) may be used to achieve this speed 80% port speed requirement if necessary. However, if multiple connections are required to achieve the 80% port speed

requirement, it should be noted. Moreover the TCP stack may need to be optimized in some fashion to achieve maximum speed. If the TCP parameters need to be changed from their default values to meet the service tier speed, than these parameter settings should be noted.

[2.1.2] The device shall support the download and upload network throughput tests (i.e. .DownloadDiagnostics. and .UploadDiagnostics. objects) and collection of all results variables defined in Table 1 of TR-143 using the FTP protocol.

[2.1.2.1] The device must be capable of performing a FTP download test at a speed equal to or exceeding 80% of the port speed (e.g. 80Mbps on a 100Mbps port, 800Mbps on a GigE port, etc …) at 20 msec round trip time. Multithreaded operation (i.e. multiple TCP connections) may be used to achieve this speed 80% port speed requirement if necessary. However, if multiple connections are required to achieve the 80% port speed requirement, it should be noted. Moreover the TCP stack may need to be optimized in some fashion to achieve maximum speed. If the TCP parameters need to be changed from their default values to meet the service tier speed, than these parameter settings should be noted.

[2.1.2.2] The device must be capable of performing a FTP upload test at a speed equal to or exceeding 80% of the port speed (e.g. 80Mbps on a 100Mbps port, 800Mbps on a GigE port, etc …) at 20 msec round trip time. Multithreaded operation (i.e. multiple TCP connections) may be used to achieve this speed 80% port speed requirement if necessary. However, if multiple connections are required to achieve the 80% port speed requirement, it should be noted. Moreover the TCP stack may need to be optimized in some fashion to achieve maximum speed. If the TCP parameters need to be changed from their default values to meet the service tier speed, than these parameter settings should be noted.

[2.1.3] The device must support the UDP EchoPlus test (i.e. .UDPEchoConfig. Object) and collection of all results variables defined in Table 1 of TR-143. Note the device shall operate as a UDPEcho server (reflector) when .UDPEchoConfig.Enable is True, and will function as a UDPEchoPlus server when .UDPEchoConfig.Enable is True and .UDPEchoConfig.EchoPlus Enabled is True.

[2.1.4] As defined in TR-143, the device shall perform a single HTTP (or FTP) download throughput test when the .DownloadDiagnostics.DiagnosticState variable in Table 1 is set to the string “Requested”. The status of that test will be placed in .DownloadDiagnostics.DiagnosticState upon completion of the test.

[2.1.5] As defined in TR-143, the device shall perform a single HTTP (or FTP) upload throughput test when the UploadDiagnostics.DiagnosticState variable in Table 1 is set to the string “Requested”. The status of that test will be placed in UploadDiagnostics.DiagnosticState upon completion of the test.

[2.1.6] The device shall be capable of performing a HTTP/FTP throughput test simultaneously while functioning as a UDPEcho or UDPEchoPlus server.

[2.1.7] The device shall support all of the functionality defined in this document when operating in PPPoE mode for High Speed Internet (HSI) service.

2.2 Network Throughput Tests Theory of Operation Summary

This section summarizes the theory of operation for the network throughput tests. It illustrates the procedures followed to perform tests and populate the corresponding results variables in Table 1 of TR-143. Each throughput test path (Download and Upload) and test protocol (HTTP and FTP) is taken in turn. We begin by providing the throughput test section of Table 1 in TR-143.

Name[1]	Type	Write[2]	Description	Object Default
.Capabilities. .Capabilities.-PerformanceDiagnostic	object	- -	The capabilities of the device. This is a constant read-only object, meaning that only a firmware upgrade will cause these values to be altered. The capabilities of the Performance Diagnostics (DownloadDiagnostics and UploadDiagnostics) for the device.	-
DownloadTransports	string	-	Comma-separated list of supported DownloadDiagnostics transport protocols for a CPE device. Each item in the list is an enumeration of: “HTTP” “FTP” (OPTIONAL)	-
UploadTransports	string	-	Comma-separated list of supported UploadDiagnostics transport protocols for a CPE device. Each item in the list is an enumeration of: “HTTP” “FTP” (OPTIONAL)	-
.DownloadDiagnostics.	object	-	This object defines the diagnostics configuration for a HTTP and FTP DownloadDiagnostics Test. Files received in the DownloadDiagnostics do not require file storage on the CPE device.	-
DiagnosticsState	string	W	Indicate the availability of diagnostic data. One of: “None” “Requested” “Completed” “Error_InitConnectionFailed” “Error_NoResponse ” “Error_TransferFailed” “Error_PasswordRequestFailed” “Error_LoginFailed” “Error_NoTransferMode”	-

			“Error_NoPASV” “Error_IncorrectSize” “Error_Timeout” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate the corresponding diagnostic test. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the test is completed, the value of this parameter MUST be either Completed (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Completed, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	Specifies the IP-layer interface over which the test is to be performed. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE MUST use the default routing interface.	-
DownloadURL	string(256)	W	The URL as defined in [3], for the CPE to perform the download on. This parameter MUST be in the form of a valid HTTP [2] or FTP [6] URL. When using FTP transport, FTP binary transfer MUST be used. When using HTTP transport, persistent connections MUST be used and pipelining MUST NOT be used.	-

			When using HTTP transport the HTTP Authentication MUST NOT be used.	
DSCP	unsignedInt [0:63]	W	The DiffServ code point for marking packets transmitted in the test. The default value SHOULD be zero.	-
EthernetPriority	unsignedInt [0:7]	W	Ethernet priority code for marking packets transmitted in the test (if applicable). The default value SHOULD be zero.	-
ROMTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the client sends the GET command. For FTP this is the time at which the client sends the RTRV command.	-
BOMTime	dateTime	-	Begin of transmission time in UTC, which MUST be specified to microsecond precision For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the first data packet is received. For FTP this is the time at which the client receives the first data packet on the data connection.	-
EOMTime	dateTime	-	End of transmission in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the last data packet is received. For FTP this is the time at which the client receives the last packet on the data connection.	-
TestBytesReceived	unsignedInt	-	The test traffic received in bytes during the FTP/HTTP transaction including FTP/HTTP headers, between BOMTime and EOMTime,	-
TotalBytesReceived	unsignedInt	-	The total number of bytes received on the Interface between BOMTime and EOMTime.	-
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP socket open (SYN) was sent for the HTTP connection. For FTP this is the time at which the TCP socket open (SYN) was sent for the data connection. Note: Interval of 1 microsecond SHOULD be supported.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP ACK to the socket opening the HTTP connection was received. For FTP this is the time at which the TCP ACK to the socket opening the data connection was received. Note: Interval of 1 microsecond SHOULD be supported.	-
.UploadDiagnostics.	object	-	This object defines the diagnostics configuration for a HTTP or FTP UploadDiagnostics test. Files sent by the UploadDiagnostics do not require file storage on the CPE device, and MAY be an arbitrary stream of bytes	--
DiagnosticsState	string	W	Indicate the availability of diagnostic data. One of: “None”	-

			“Requested” “Completed” “Error_InitConnectionFailed” “Error_NoResponse” “Error_PasswordRequestFailed” “Error_LoginFailed” “Error_NoTransferMode” “Error_NoPASV” “Error_NoCWD” “Error_NoSTOR” “Error_NoTransferComplete” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate the corresponding diagnostic test. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the test is completed, the value of this parameter MUST be either Completed (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Completed, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	IP-layer interface over which the test is to be performed. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value.	-

			If an empty string is specified, the CPE MUST use the default routing interface.	
UploadURL	string(256)	W	The URL as defined in [3], for the CPE to Upload to. This parameter MUST be in the form of a valid HTTP [2] or FTP [6] URL. When using FTP transport, FTP binary transfer MUST be used. When using HTTP transport, persistent connections MUST be used and pipelining MUST NOT be used. When using HTTP transport the HTTP Authentication MUST NOT be used.	-
DSCP	unsignedInt [0:63]	W	DiffServ code point for marking packets transmitted in the test. The default value SHOULD be zero.	-
EthernetPriority	unsignedInt [0:7]	W	Ethernet priority code for marking packets transmitted in the test (if applicable). The default value SHOULD be zero.	-
TestFileLength	unsignedInt	W	The size of the file (in bytes) to be uploaded to the server. The CPE MUST insure the appropriate number of bytes are sent.	-
ROMTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the client sends the PUT command For FTP this is the time at which the STOR command is sent.	-
BOMTime	dateTime	-	Begin of transmission time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the first data packet is sent. For FTP this is the time at which the client receives the ready for transfer notification.	-
EOMTime	dateTime	-	End of transmission in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time when the HTTP successful response code is received. For FTP this is the time when the client receives a transfer complete.	-
TotalBytesSent	unsignedInt	-	The total number of bytes sent on the Interface between BOMTime and EOMTime.	-
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP socket open (SYN) was sent for the HTTP connection. For FTP this is the time at which the TCP socket open (SYN) was sent for the data connection Note: Interval of 1 microsecond SHOULD be supported.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the Time at which the TCP ACK to the socket opening the HTTP connection was received.	-

			For FTP this is the Time at which the TCP ACK to the socket opening the Data connection was received. Note: Interval of 1 microsecond SHOULD be supported.	

Table 1 (Table 1 of TR-143)

Notes:

(1) The purpose of the throughput test is to measure the available bandwidth of the service. Towards that end, the test should be capable of meeting or exceeding a throughput equivalent to the FiOS service tier (see requirements 2.1.1.1 through 2.1.2.2). Multiple threading or large widow sizes and corresponding receiver buffering may be required to meet the speeds of the higher service tiers. In section 3, a variable is defined to indicate when multiple threads are being used.

(1) While TestBytesReceived measures the length of the test file corresponding to the HTTP (or FTP) download between ROM time and EOM time, TotalBytesReceived is the total traffic measured on the interface (e.g. WAN port) on which the test is run (between ROM time and EOM time). The difference in these two values provides some indication as the amount of subscriber traffic (e.g. VOD) that occurred during the test.
(2) TCPOpenRequestTime is a timestamp of the SYN packet used to set of the test connection. TCPOpenResponseTime is a timestamp of the SYN-ACK packet sent from the test server in response to the SYN. These values provide an estimate of the TCP Round Trip Time to the test server.

The following figures (Figure 6 and Figure 7 of TR-143) depict the procedures for the Download and Upload throughput test, respectively, using HTTP protocol. In particular, the figures illustrate how the ROM, BOM, and EOM variables are set as a result of a test. Additionally, the TestBytesReceived, TotalBytesReceived, TCPOpenRequestTime, TCPOpenResponseTime, and Diagnostic State variables must be set upon completion of a test. The configuration variables are Interface, Download URL, DSCP setting and, EthernetPriority setting. Normally the test is triggered by a TR-069 command which sets the DiagnosticState to value of “Requested”.

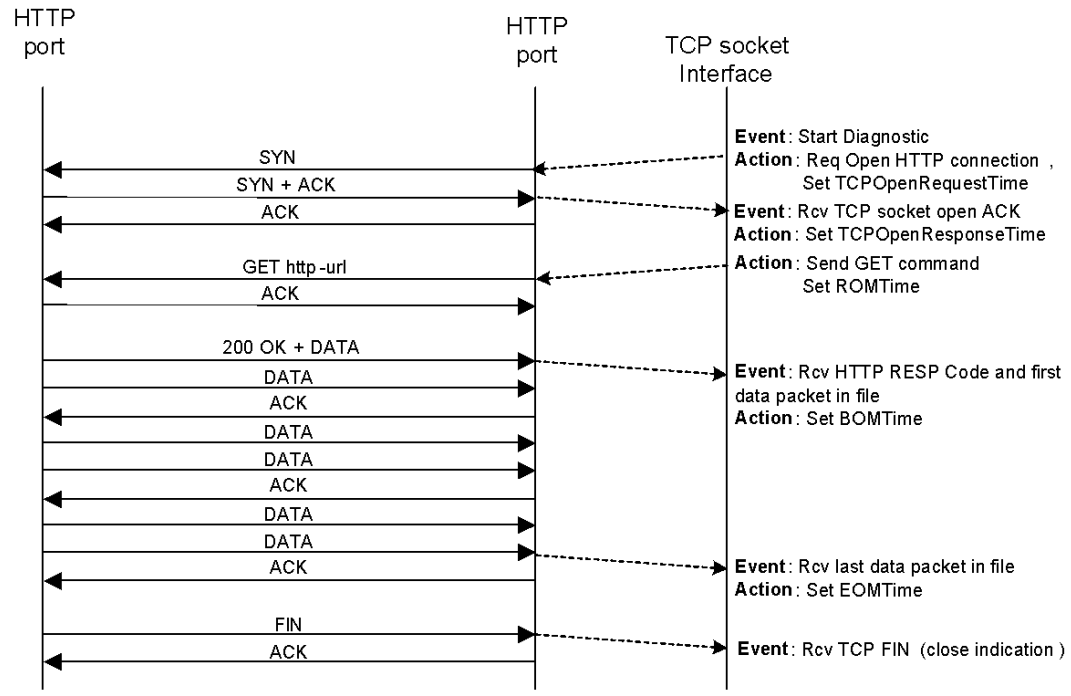

Figure SEQ Figure * ARABIC 1 (Figure 6 of TR-143 – HTTP Download Test)

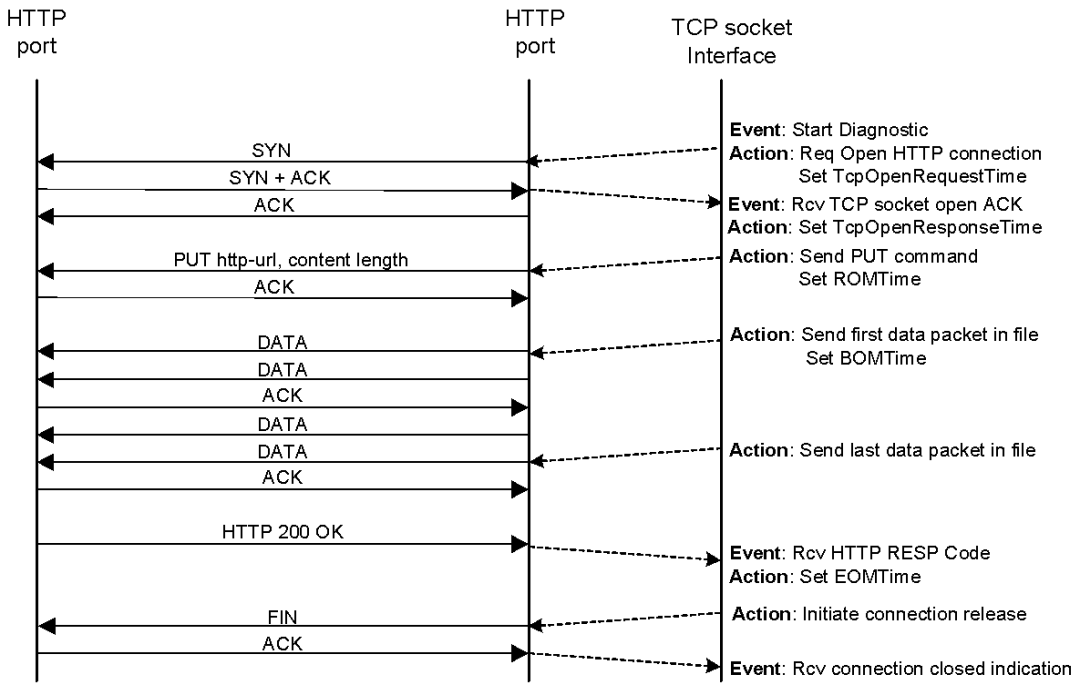

Figure 2 (Figure 7 of TR-143 – HTTP Upload Test)

2.3 UDP Echo Plus Test Theory of Operation Summary

This section summarizes the theory of operation for the UDP Echo Plus test. The UDP Echo Plus test extends UDP Echo by incorporating additional packet fields and client/server behaviors. We begin by providing the UDP Echo Plus test section of Table 1 in TR-143.

Name	Type	Write	Description	Object Default
.UDPEchoConfig.	object	-	This object allows the CPE to be configured to perform the UDP Echo Service defined in [4] and UDP Echo Plus Service defined in Appendix A.1.	-
Enable	boolean	W	MUST be enabled to receive UDP echo. When enabled from a disabled state all related timestamps, statistics and UDP Echo Plus counters are cleared.	-
Interface	string(256)	W	IP-layer interface over which the CPE MUST listen and receive UDP echo requests on. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value.	-

			If an empty string is specified, the CPE MUST listen and receive UDP echo requests on all interfaces. Note: Interfaces behind a NAT MAY require port forwarding rules configured in the Gateway to enable receiving the UDP packets.	
SourceIPAddress	string	W	The Source IP address of the UDP echo packet. The CPE MUST only respond to a UDP echo from this source IP address.	-
UDPPort	unsignedInt	W	The UDP port on which the UDP server MUST listen and respond to UDP echo requests.	-
EchoPlusEnabled	boolean	W	If True the CPE will perform necessary packet processing for UDP Echo Plus packets.	-
EchoPlusSupported	boolean	-	True if UDP Echo Plus is supported.	-
PacketsReceived	unsignedInt	-	Incremented upon each valid UDP echo packet received.	-
PacketsResponded	unsignedInt	-	Incremented for each UDP echo response sent.	-
BytesReceived	unsignedInt	-	The number of UDP received bytes including payload and UDP header after the UDPEchoConfig is enabled.	-
BytesResponded	unsignedInt	-	The number of UDP responded bytes, including payload and UDP header sent after the UDPEchoConfig is enabled.	-
TimeFirstPacketReceived	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The time that the server receives the first UDP echo packet after the UDPEchoConfig is enabled.	-
TimeLastPacketReceived	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 The time that the server receives the most recent UDP echo packet.	-

Table 2 (UDP Echo Plus test section of Table 1 in TR-143)

The UDP Echo Plus packet contains the packet fields specified in Figure 3 (UDP Echo Plus Packet Format) below. Each field is 4 bytes (32 bits) except for the TestRespRecvDateTimeStr* field which is 32 bytes long. The TestRespRecvTimeStamp and TestRespReplyTimeStamp timestamps each require a continuous wrapping 32 bit (Big Endean) counter that begins on startup and counts in microseconds. In Figure 3 (UDP Echo Plus Packet Format), five additional fields are added beyond the standard (TestRespReplyIncomingDiscards*, TestRespReplyOutgoingDiscards*, TestRespRecvDateTimeStr*, TestRespRecvIncomingBufferOcc*, and TestRespRecvOutgoingBufferOcc*) for experimental purposes. Note that TestRespRecvDateTimeStr* is the same timestamp as TestRespRecvTimeStamp but provides the absolute date-time value for TestRespRecvTimeStamp in string format. The TestRespRecvDateTimeStr* is a 32 byte date-time string having the form "2008-04-09T15:01:05.123456" representing the date-time of the UDPEchoPlus server (whether BHR or test server) to the microsecond (in GMT). Note that TestRespRecvDateTimeStr* provides the absolute date-time reference required to infer one-way delays between the test server and BHR (when used with a Network Time Protocol synchronized test server). TestRespRecvIncomingBufferOcc* and TestRespRecvOutgoingBufferOcc* denote the queue occupancies of the ingress queue (upon the UDPEchoPlus packet arrival) and egress queue (upon the UDPEchoPlus departure) for the device serving as the UDPEchoPlus server/reflector during the test

(whether BHR or test server). Note that TestRespRecvIncomingBufferOcc* and TestRespRecvOutgoingBufferOcc* apply to the queues on the port that the UDPEchoPlus test is run on (e.g. BHR WAN Port).

When a UDPEchoPlus capable server receives a standard UDP Echo packet with insufficient size to support the standard UDPEchoPlus fields (i.e. atleast 20 bytes of payload), the UDPEchoPlus server just reflects the request back towards the source with no payload modification. When UDP Echo request packets are larger than the minimal packet length for UDPEchoPlus packets, the first 20 bytes (or 28 bytes when using the experimental fields) of payload are filled as described in this section. Note that the TestRespRecvDateTimeStr* field is only included when the UDPEchoPlus packet payload is greater than or equal 60 bytes though the other UDPEchoPlus fields must be populated whenever the payload is greater than or equal to 20 bytes (or 28 bytes when using the experimental fields) . So when UDPEchoPlus is enabled, all UDPEchoPlus fields other than TestRespRecvDateTimeStr* will be populated in UDP Echo packets having payloads between 20 bytes (or 28 bytes when using the experimental fields) and 59 bytes and TestRespRecvDateTimeStr* will be populated when the UDP Echo payload is greater than or equal to 60 bytes. Similarly, the buffer occupancy fields are populated when the payload size is greater than or equal to 68 bytes.

Source Port	Destination Port
Length	Checksum
TestGenSN	
TestRespSN	
TestRespRecvTimeStamp	
TestRespReplyTimeStamp	
TestRespReplyFailureCount	
TestRespReplyIncomingDiscards *	
TestRespReplyOutgoingDiscards *	
TestRespRecvDateTimeStr* (32 bytes)	
TestRespRecvIncomingBufferOcc*	
TestRespRecvOutgoingBufferOcc*	

Figure 3 (UDP Echo Plus Packet Format)

- **TestGenSN** – The packet's sequence number set by the UDP client in the echo request packet, and is left unmodified in the response. It is set to an initial value upon the first requests and incremented by 1 for each echo request sent by the UDP client.
- **TestRespSN** – The UDP Echo server's count that is incremented upon each valid echo request packet it receives and responded to. This count is set to 0 when the UDP Echo server is enabled and incremented by one for each UDP Echo Plus packet that is returned to the client.
- **TestRespRecvTimeStamp** is set by the UDP Echo Plus server to record the reception time of echo request packet and is sent back to the server in this data field of the echo response packet.

- **TestRespReplyTimeStamp** is set by the UDP Echo Plus server to record the forwarding time of the echo response packet.
- **TestRespReplyFailureCount** is set by the UDP Echo Plus server to record the number of locally dropped echo response packets. This count is incremented if a valid echo request packet is received but for some reason can not be responded to (e.g. due to local buffer overflow, CPU utilization, etc...). It is a cumulative count with its current value placed in all request messages that are responded to. This count is set to 0 when the UDP Echo server is enabled.
- **TestRespReplyIncomingDiscards** – the number of discards measured on the incoming interface of the interface port of the UDP Echo plus server at the time the UDP Echo plus packets is received.
- **TestRespReplyOutgoingDiscards** – the number of discards measured on the outgoing interface of the interface port of the UDP Echo plus server at the time the UDP Echo plus packets is sent (i.e. returned to the client).
- **TestRespRecvDateTimeStr** – denotes the absolute time (and date) of the TestRespRecvTimestamp in (32 byte) string format. TestRespRecvDateTimeStr is a 32 byte string having the form "2008-04-09T15:01:05.123456" representing the date-time equivalent of TestRespRecvTimeStamp for the BHR to the microsecond (in GMT). It is used to provide an absolute date/time reference for the 32 bit TestRespRecvTimeStamp value. TestRespRecvDateTimeStr is only included in UDP Echo Plus packets having a payload greater than or equal to 60 bytes.
- **TestRespRecvIncomingBufferOcc -** the occupancy in number of bytes measured at the ingress queue of incoming interface port of the UDPEcho Plus server at the time the UDP Echo plus packets is received. Note, if the device does not have an ingress queue on the incoming WAN interface, then this field should be set to zero. ~~Alternatively it may show the value of the LAN side egress queue having the largest value at the time the UDP Echo plus packets is received. It is expected LAN egress queues could reach non-zero values due to wireless or other bandwidth fluctuations in the LAN.~~ TestRespRecvIncomingBufferOcc is only included in UDP Echo Plus packets having a payload greater than or equal to 64 bytes.
- **TestRespRecvOutgoingBufferOcc -** the occupancy in number of bytes measured at the egress queue of outgoing interface port of the UDP Echo Plus server at the time the UDP Echo plus response packet is sent (i.e. returned to the client). TestRespRecvOutgoingBufferOcc is only included in UDP Echo Plus packets having a payload greater than or equal to 68 bytes.

UDP Echo and UDP Echo Plus server setup.
When enabled the UDP Echo server will accept UDP Echo packets. When enabled the UDPEchoPlus server will accept UDP Echo Plus packets and perform the specific operation in the following sections. When UDP Echo server is enabled all counters are reset to 0.

UDP Echo Plus Client
The UDP Echo client sends packets that match the UDP Echo Plus server configuration (Destination and Source IP address, UDP port). The UDP Echo Plus client controls the provisional settings such as DHCP code point settings, packet size and inter-arrival spacing. The UDP Echo Plus client will place a 32 bit sequence number value, which increments by 1 for each request packet sent, in the TestGenSN field of each UDP Echo Plus request.

Note #1: The counter will roll over every 71.5 minutes. Every two successive UDP Echo Plus packets used in the same test would need to be sent within that interval for jitter computation (loss computation of course would not have this restriction).

Note #2: Devices that don't support 1usec timestamp resolution, will still compute the timestamp in microseconds, providing a multiplier. For example a device with 1msec resolution, will increment the 32bit timestamp field in steps of 1000 (instead of steps of 1).

Note #3: The incoming and outgoing discards experimental fields are used to detect local congestion at the UDP Echo server (i.e. BHR) at the time UDP Echo Plus packets are processed. UDP Echo Plus packets dropped at the UDP Echo Plus server should not factor into packet loss measurements of the network.

Note #4: In practice, the BHR functions as the UDP Echo server and the network test server functions as the UDP Echo Plus client.

3. Extended Functional Requirements for VZ

3.1. Throughput Test Initiation Mechanism (Phase 1)

This section describes a custom mechanism for the initiation of download and upload throughput tests from a moderate population (<=5000) of subscribers for the purposes of network performance benchmarking. As such, these requirements (Extended Functional Requirements) are distinct from the standard TR-143 functional requirements presented in Section 2 and are presented in a separate section. However, some of the methods described (or its components) provide the basis for a BBF contribution on this aspect. Each of the variables described here are included in the path: InternetGatewayDevice.X_ACTIONTEC_PerformanceDiagnostics.

[3.1.1] The device must support the following test configuration variables (in the path - InternetGatewayDevice.X_ACTIONTEC_PerformanceDiagnostics.): ThroughputTestingEnabled (bool), PerformDownloadThroughputTest (bool), PerformUploadThroughputTest (bool), TestServerIPAddress (string), TestRequestInterval (unsignedInt – in seconds), TestRequestUDPPort (unsignedInt), TestRequestTimeout (unsignedInt – in milliseconds), TestResultsTCPPort (unsignedInt) , LowActivityThreshold (unsignedInt – bits/sec), LowActivityWaitTime (unsignedInt – in seconds).

[3.1.2] When testing is enabled (ThroughputTestingEnabled = True), the device must send a TestRequest packet to the test server periodically (with constant period of once every TestRequestInterval) with the initial request starting at a (uniformly distributed) random time (to provide an offset among the BHR tests requests). Alternatively, the BHR may send a TestRequest at a random time within each TestRequestInterval. *Each Home Router (BHR) must be assigned a unique random seed for the (uniformly distributed) random time selection.* Hence, the tests requests are recurring with one attempt every TestRequestInterval (either periodically with constant period and random initial time, or at a random time in each interval). The TestRequest packets are sent to the UDP Port TestRequestUDPPort of the test server which has IP address TestServerIPAddress. The TestRequest packet is a UDP packet having the payload format depicted in Figure 6.

[3.1.3] A BHR may send its next TestRequest packet starting from its scheduled random time epoch (per requirement [3.1.2]) but after a LowActivityWaitTime seconds have passed with the WAN port utilization having a value less than LowActivityThreshold (bits/sec). The purpose of this requirement is to trigger tests during periods of low to no subscriber activity.

[3.1.4] If a TestResponse[1] packet is received (upon being returned from the test server) with DTE = 1, than a download throughput test is performed using the configuration parameters of Table 1. The protocol used to perform the download test is defined in the RequestedTestProtocol variable.

[3.1.5] If a Test Response packet is received (upon being returned from the test server) with UTE = 1, than an upload throughput test is performed using the configuration parameters of Table 1. The protocol used to perform the upload test is defined in the RequestedTestProtocol variable.

[3.1.6] Upon completion of a download and/or upload test, a TCP connection is opened to the test server on port TestResultsTCPPort. After opening the TCP connection, a TestResult packet is sent to the test server. The TestResult packet has a 480 byte payload and has which includes the results of the throughput test. It has the following format:

[1] Note that a TestRequest packet is also referred to as a TestResponse packet when traversing back towards the BHR.

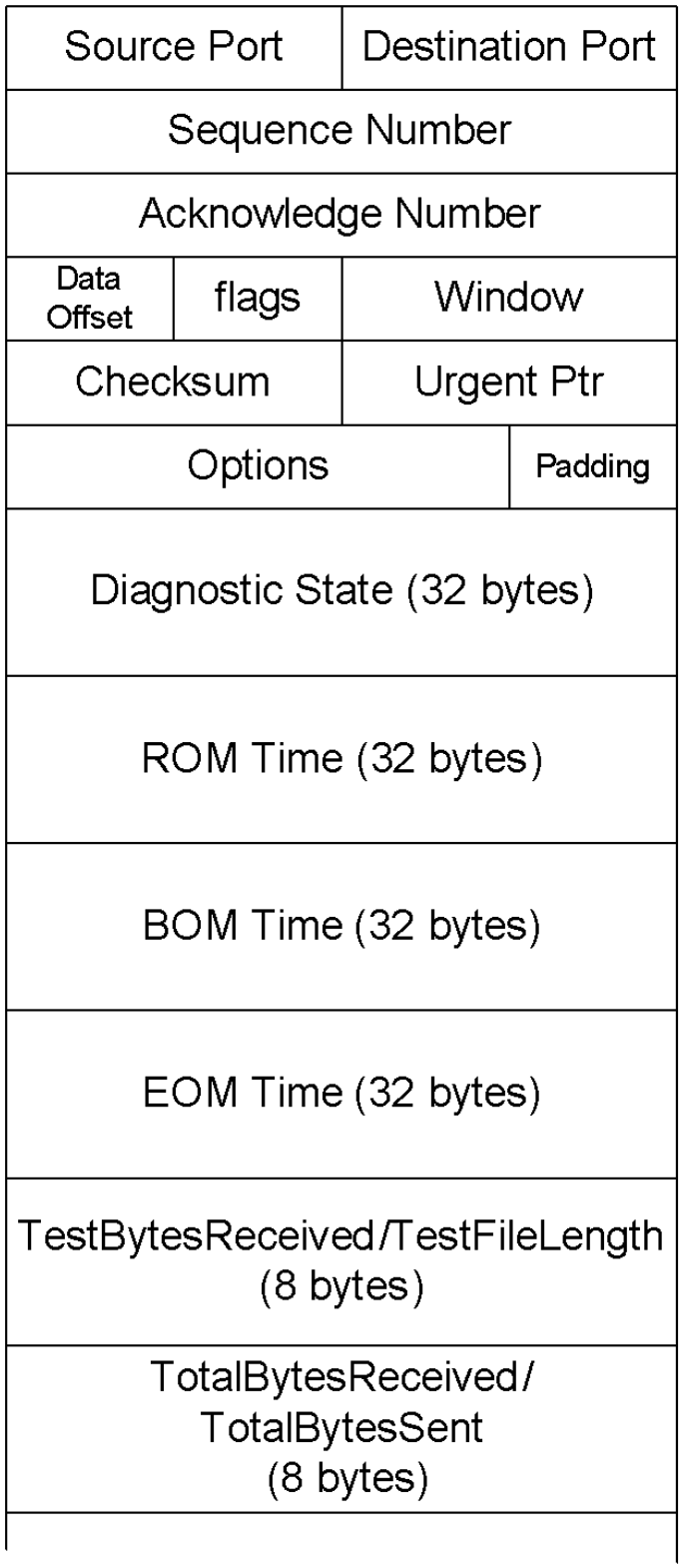
Source Port
Destination Port
Sequence Number
Acknowledge Number
Data Offset
flags
Window
Checksum
Urgent Ptr
Options
Padding
Diagnostic State (32 bytes)
ROM Time (32 bytes)
BOM Time (32 bytes)
EOM Time (32 bytes)
TestBytesReceived/TestFileLength (8 bytes)
TotalBytesReceived/ TotalBytesSent (8 bytes)

Figure 4 (TestResult packet format)

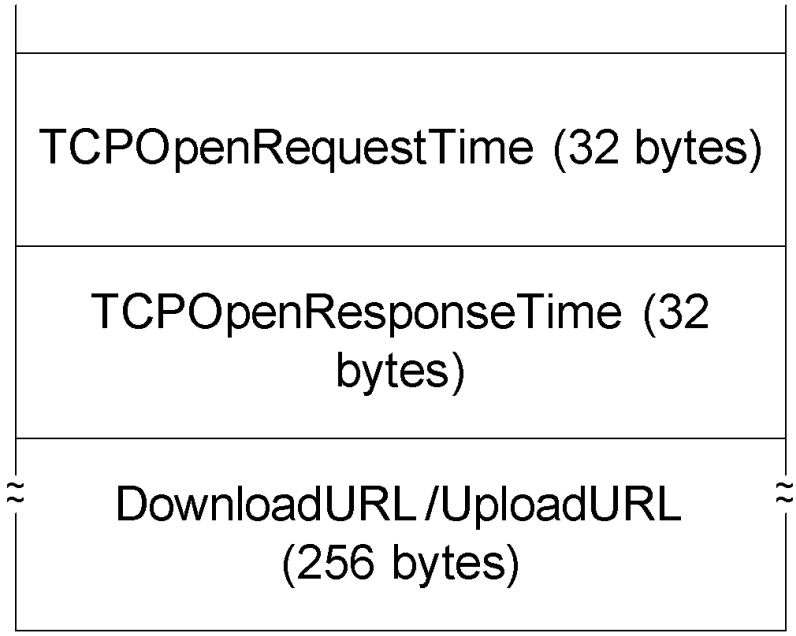

Figure 5 (TestResult packet format - continued)

The TestResult packet format is the same for a download and upload throughput test except for the shared fields: TestBytesReceived/TestFileLength (8 bytes), TotalBytesReceived/TotalBytesSent (8 bytes), and DownloadURL/UploadURL (256 bytes) which apply to a download or upload, respectively. If both the download and upload tests are performed, two separate TestResult packets are sent in the TCP connection. If only a download or upload is completed, then a single TestResult packet is sent for the appropriate result. The TestResult packet contains a DiagnosticState field (32 bytes), a ROM Time field (32 bytes), a BOM Time field (32bytes), an EOM Time field (32 bytes), a TestBytesReceived/TestFileLength (8 bytes), a TotalBytesReceived/TotalBytesSent (8 bytes), TCPOpenRequestTime (32 bytes), TCPOpenResponseTime (32 bytes), and a DownloadURL/UploadURL (256 bytes). The function and encoding of each of these fields is defined in Table 1 of TR-143. In particular, each of the timestamps ROM Time, BOM Time, EOM Time, TCPOpenRequestTime, and TCPOpenResponseTime are encoded according to TR-069

dateTime format with microsecond resolution encoding. That is, ISO 8601 with time value having microsecond resolution (e.g. 2008-04-09T15:01:05.123456*). Also, note that the TestBytesReceived/TestFileLength and TotalBytesReceived/TotalBytesSent are both 8 byte fields each having long-long integer values transmitted in Big Endian (Most Significant Byte first) format.* Using the TestResult packets the test server can construct a database of test results and begin build a performance profile of the network path from the test server to the BHR.

[3.1.7] The UDP Echo Plus capability must be able to operate simultaneously with the HTTP or FTP throughput test.

[3.1.8] The NDT protocol should be supported as a throughput test option in addition to HTTP and FTP. The NDT test must operate at a speed equal to or exceeding 80% of the port speed (e.g. 80Mbps on a 100Mbps port, 800Mbps on a GigE port, etc …) at 20 msec round trip time. Multithreaded operation (i.e. multiple TCP connections) may be used to achieve this speed 80% port speed requirement if necessary.

3.2. Throughput Test Initiation Mechanism (Phase 2)

This section describes the second phase (Phase 2) for the TR143 tests initiation protocol. Phase 2 is backward compatible with Phase 1 (which has been removed from the specification since the overlapping functionality is redundant). That is, a Phase 1 client or server will interoperate with a Phase 2 client or server. The purpose of this 2nd phase is to introduce an "NDT" test, "DNS" response test, "WebPage" response test, Reverse UDP Echo Plus test, "Two Way Active Measurement Protocol" (TWAMP) test, "PathPing" test, "Passive Usage Monitoring" test, UDPThroughput, and to generalize the HTTP test to support multiple connections and a time-based test mode. In addition, the test server will also be able to re-direct throughput tests to different URLs, and to provide more control for balancing the load on the test server (by introducing explicit times for generation of the next TestRequest packets by the BHR's). Phase 2 assumes the test "control" (i.e. TestResquest/TestResponse protocol) function of the server is distinct from (i.e. can be at a different physical location then) the test "execution" (the act of performing actual throughput test). As such, the URL that the test will be run to is provided in the TestResponse packet (and not assumed to be the same location that the TestRequest packets are sent to). Note that for NDT, the NDTServerIPaddress and NDTServerTCPport of the NDT server are used to construct a custom URL since an IP address and TCP port are all that is required for an NDT test. Also, a new value for the DiagnosticState is supported. In addition to having the ability to set DiagnosticState to the string value of "Requested", the DiagnosticState parameter can also be set to "Request". The difference being, when set to "Requested" the configured test is run

immediately without test control but when set to “Request”, the configured test is run with test control. That is, “Request” triggers the device to immediately send a TestRequest packet independent of the TestRequestInterval timing.

[3.2.1] The device must support the Phase 1 test configuration variables listed hin [3.1.1]. In addition, the device must support the following additional Phase 2 variables (included in the path - InternetGatewayDevice.X_ACTIONTEC_PerformanceDiagnostics.): TestRequestLimit (unsignedInt), SpeedTierScaling (unsignedInt), EnableTLSProtectedTestResults (Boolean), LowActivityWaitExpirationTime (unsignedInt - seconds), LowActivityWaitExpirationCount (unsignedInt), DownstreamSpeedTier (string[16] - Kbps), UpstreamSpeedTier (string[16] - Kbps), DeviceSerialNumber (string[64]), LATA_ID (string[64]), GWR_ID (string[64]), OLT_ID(string[64]), ONT_ID(string[64]). The full list of Phase 2 (and Phase 1) test configuration variables is depicted in the following table:

Name[1]	Type	Write[2]	Description	Object Default
InternetGatewayDevice.X_ACTIONTEC_PerformanceDiagnostics.	object	-	This object allows the CPE to be configured to perform the TR-143 test control protocol	-
DownloadTransports	string	-	Comma-separated list of supported DownloadDiagnostics transport protocols for a CPE device. Each item in the list is an enumeration of: “HTTP” “FTP” “NDT” “DNS” “WebPage” “ReverseUDPEchoPlus” “TWAMP” “PathPing” “PassiveUsageMonitoring” “UDPThroughput”	-
UploadTransports	string	-	Comma-separated list of supported UploadDiagnostics transport protocols for a CPE device. Each item in the list is an enumeration of: “HTTP” “FTP” “UDPThroughput”	-
ThroughputTestingEnabled	bool	W	True if Throughput testing is enabled	-
PerformDownloadThroughputTest	bool	W	True if Download Throughput testing is enabled	-
PerformUploadThroughputTest	bool	W	True if Upload Throughput testing is enabled	-

TestServerIPAddress	string	W	IP address to test server (to send TestRequest messages and TestResult messages to)	-
TestRequestInterval	unsigned int (seconds)	W	A recurring fixed time interval within which a TestRequest is made to the test server. The TestRequestInterval repeats until ThroughputTestingEnabled is set to false	-
TestRequestLimit	unsigned int	W	The number of periodic tests to request before automatically setting ThroughputTestingEnabled to FALSE and no longer running periodic tests (until ThroughputTestingEnabled is re-enabled). This parameter allows the CPE to perform a fixed number of requested tests before no longer sending TestRequest packets. When this parameter is unassigned, or set to zero, then it does not take affect and periodic tests run indefinitely.	-
SpeedTierScaling	unsigned int	W	This parameter is an unsigned integer used to set the scaling for speedtier values placed in the SpeedTier field of TestRequest packets.The value placed in each SpeedTier subfield (DownstreamSpeedTier subfield and UpstreamSpeedTier subfield in the test request message) is determined by first converting the corresponding parameter (DownstreamSpeedTier and UpstreamSpeedTier) to an unsigned integer representation, ignoring the units (e.g. Mbps, Kbps, etc …) and then placing the integer part that value divided by this parameter. Note that if the DownstreamSpeedTier or UpstreamSpeedTier value is zero or, empty or unassigned, then the corresponding DownstreamSpeedTier subfield and UpstreamSpeedTier subfield in the test request message is set to zero.	
TestRequestUDPPort	unsigned int	W	UDP port on which each TestRequest is made (i.e. UDP	-

			port value of TestRequest packets)	
TestRequestTimeout	unsigned int (milliseconds)	W	The amount of time a BHR waits for receiving a TestResponse packet in response to a TestRequest packet is has sent to the test server	-
TestResultsTCPPort	unsigned int	W	TCP port on which a TestResult packets are sent (i.e. TCP port value of the TCP connection used to send TestResult packet)	-
EnableTLSProtectedTestResults	boolean	W	Enables Transport Layer Security for TCP connection used to forward tests results (i.e. TestResult packets).	-
LowActivityThreshold	unsigned int (bit/sec)	W	A bit rate threshold which must not be exceeded for LowActivityWaitTime milliseconds before a TestRequest messge can be sent when Wait For Idle (WFI) mode is enabled	-
LowActivityWaitTime	unsigned int (milliseconds)	W	The amount of time for which LowActivityThreshold can not be exceeded on the WAN port before sending a TestRequest message when Wait For Idle (WFI) mode is enabled	-
LowActivityWaitExpirationTime	unsigned int (seconds)	W	The amount of time to wait for the LowActivityThreshold to not be exceeded for LowActivityWaitTime before giving up on the test for the current TestRequestInterval	-
LowActivityWaitExpirationCount	unsigned int	W	A count of the number of times a LowActivityWaitExpirationTime exceeded event occurred since ThroughputTestingEnabled was set	-
ClientVersionNumber	unsigned int [0:7]	W	This version number placed on the Version field bits in the TestRequest packets. E.g. when ClientVersionNumber = 0 then Ver2 = 0, Ver1 = 0, Ver0 = 0, or when ClientVersionNumber = 1 then Ver2 = 0, Ver1 = 0, Ver0 = 1, etc …	-
RequestedTestProtocol	String		A string value denoting the test protocol type to be set in the	-

			Protocol Type (PT) bits in each test request message. Note the actual test that is run is defined by the value of the PT bits returned from the test server. However, the parameter indicates which test protocol is requested. It can be set to one of the following string values: “HTTP” or “FTP” or “NDT” or “DNS” or “WebPage” or “ReverseUDPEchoPlus or TWAMP” or “PathPing” or “PassiveUsageMonitoring”	
NDTClientVersion	String	W	String denoting the version of the NDT client to use for performing NDT tests	-
DownstreamSpeedTier	String[16]	W	A string denoting the downstream speed tier in Kbps (e.g. 25000 denotes a 25Mbps downstream tier)	-
UpstreamSpeedTier	String[16]	W	A string denoting the upstream speed tier in Kbps (e.g. 25000 denotes a 25Mbps upstream tier)	-
DeviceSerialNumber	String[64]	W	The serial number of the current device (e.g. the Home Gateway device serial number)	-
LATA_ID	String[64]	W	A string designation for the LATA the device is in	-
GWR_ID	String[64]	W	A string designation for the serving GWR for this device	-
OLT_ID	String[64]	W	A string designation for the serving OLT for this device	-
ONT_ID	String[64]	W	A string designation for the serving ONT (or ONT model type) for this device	-
.DownloadDiagnostics.	Object	W	This object defines the diagnostics configuration for a HTTP and FTP DownloadDiagnostics Test.Files received in the DownloadDiagnostics do not require file storage on the CPE device.	-
DiagnosticsState	string	W	Indicate the availability of diagnostic data. One of: “None” “Request” “Requested” “Completed”	-

			“Error_InitConnectionFailed” “Error_NoResponse ” “Error_TransferFailed” “Error_PasswordRequestFailed” “Error_LoginFailed” “Error_NoTransferMode” “Error_NoPASV” “Error_IncorrectSize” “Error_Timeout” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate the corresponding diagnostic test. When writing, the only allowed values are Requested or Request. When set to Request, a TestRequest packet is sent and the test performed in accordance to the TestResponse packet returned from the test server. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Request or Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the test is completed, the value of this parameter MUST be either Completed (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Completed, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result	

			in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	Specifies the IP-layer interface over which the test is to be performed. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE MUST use the default routing interface.	-
DownloadURL	string(256)	W	The URL as defined in [3], for the CPE to perform the download on. This parameter MUST be in the form of a valid HTTP [2] or FTP [6] URL. When using FTP transport, FTP binary transfer MUST be used. When using HTTP transport, persistent connections MUST be used and pipelining MUST NOT be used. When using HTTP transport the HTTP Authentication MUST NOT be used.	-
DSCP	unsignedInt[0:63]	W	The DiffServ code point for marking packets transmitted in the test. The default value SHOULD be zero.	-
EthernetPriority	unsignedInt[0:7]	W	Ethernet priority code for marking packets transmitted in the test (if applicable). The default value SHOULD be zero.	-
ROMTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the client sends the GET command. For FTP this is the time at which the client sends the RTRV command.	-
BOMTime	dateTime	-	Begin of transmission time in UTC, which MUST be specified to microsecond precision For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the first data packet is received. For FTP this is the time at which the client receives the first data packet on the data connection.	-
EOMTime	dateTime	-	End of transmission in UTC, which MUST be specified to microsecond precision.	-

			For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the last data packet is received. For FTP this is the time at which the client receives the last packet on the data connection.	
TestBytesReceived	unsignedInt	-	The test traffic received in bytes during the FTP/HTTP transaction including FTP/HTTP headers, between BOMTime and EOMTime (or the first BOMTime and last EOMTime for a multi-threaded test).	-
TotalBytesReceived	unsignedInt	-	The total number of bytes received on the Interface between BOMTime and EOMTime (or the first BOMTime and last EOMTime for a multi-threaded test).	-
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP socket open (SYN) was sent for the HTTP connection. For FTP this is the time at which the TCP socket open (SYN) was sent for the data connection. Note: Interval of 1 microsecond SHOULD be supported.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP ACK to the socket opening the HTTP connection was received. For FTP this is the time at which the TCP ACK to the socket opening the data connection was received. Note: Interval of 1 microsecond SHOULD be supported.	-
ConcurrentSessions	unsignedInt	-	The number of concurrent connections to use in a multi-threaded test	-
TestBytesReceivedUnderFullLoading	unsignedInt	-	Test traffic received by the CPE between the last BOMTime and first EOMTime across all connections in a multi-threaded test.	-
TotalBytesReceivedUnderFullLoading	unsignedInt	-	The total traffic received in bytes between the last BOMTime and first EOMTime across all connections in a multi-threaded test.	-
TimeBasedTestIncrements	unsignedInt	-	When TimeBasedTestIncrements > 0, FTP/HTTP transactions are time segmented into subintervals (i.e. time slots) , each having a duration of TimeBasedTestIncrements (seconds). Defaults to zero, which implies time segmenting is disabled. TimeBasedTestIncrements is in units of seconds.	-

TimeBasedTestIncrementsOffset	unsignedInt	-	Normally TimeBasedTestIncrements segments HTTP/FTP transactions into subintervals of duration TimeBasedTestIncrements starting from BOMTime (or the last BOMTime in a multi-threaded transaction). However, when TimeBasedTestIncrementsOffset > 0, the TimeBasedTestIncrements starts at time BOMTime (or the last BOMTime in a multi-threaded transaction) + TimeBasedTestIncrementsOffset. It is used to remove the contribution from slow start in a throughput measurement. TimeBasedTestIncrementsOffset is in units of seconds.	-
DNSResponseTotal	unsignedInt		Cumulative sum of DNS response times that occur during a web page download test (milliseconds)	
.DownloadDiagnostics.Result.{i}.	object	-	This object defines the diagnostics result for a HTTP and DownloadDiagnostics Test. Each instance represents the result of one session.	-
ROMTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the client sends the GET command. For FTP this is the time at which the client sends the RTRV command. This value is attributed specifically to connection {i} of a multi-threaded test.	-
BOMTime	dateTime	-	Begin of transmission time in UTC, which MUST be specified to microsecond precision For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the first data packet is received. For FTP this is the time at which the client receives the first data packet on the data connection. This value is attributed specifically to connection {i} of a multi-threaded test.	-
EOMTime	dateTime	-	End of transmission in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the last data packet is received. For FTP this is the time at which the client receives the last packet on the data connection. This value is attributed specifically to connection {i} of a multi-threaded test.	-
TestBytesReceived	unsignedInt	-	The test traffic received in bytes during the FTP/HTTP transaction including FTP/HTTP headers, between BOMTime and EOMTime,	-

			This value is attributed specifically to connection {i} of a multi-threaded test.	
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP socket open (SYN) was sent for the HTTP connection. For FTP this is the time at which the TCP socket open (SYN) was sent for the data connection. Note: Interval of 1 microsecond SHOULD be supported. This value is attributed specifically to connection {i} of a multi-threaded test.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP ACK to the socket opening the HTTP connection was received. For FTP this is the time at which the TCP ACK to the socket opening the data connection was received. Note: Interval of 1 microsecond SHOULD be supported. This value is attributed specifically to connection {i} of a multi-threaded test.	-
.DownloadDiagnostics. .IncrementalResults{i}	object	-	Results for time segmented throughput tests. A new object is created every TimeBasedTestIncrements	-
IncTestBytesReceived	unsignedInt	-	Incremental change in the value of TestBytesReceivedUnderFullLoading every TimeBasedTestIncrements seconds between the last BOMTime and first EOMTime for a multi-threaded test	-
IncBOMTime	dateTime	-	Empirical measurement of BOMTime every TimeBasedTestIncrements seconds	-
IncEOMTime	dateTime		Empirical measurement of EOMTime every TimeBasedTestIncrements seconds	-
.UploadDiagnostics.	object	-	This object defines the diagnostics configuration for a HTTP or FTP UploadDiagnostics test.	.-
DiagnosticsState	string	W	Indicate the availability of diagnostic data. One of: “None” “Request” “Requested” “Completed” “Error_InitConnectionFailed” “Error_NoResponse” “Error_PasswordRequestFailed” “Error_LoginFailed” “Error_NoTransferMode” “Error_NoPASV” “Error_NoCWD”	-

			“Error_NoSTOR” “Error_NoTransferComplete” If the ACS sets the value of this parameter to Request or Requested, the CPE MUST initiate the corresponding diagnostic test. When writing, the only allowed values are Requested or Request. When set to Request, a TestRequest packet is sent and the test performed in accordance to the TestResponse packet returned from the test server. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the test is completed, the value of this parameter MUST be either Completed (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Completed, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then	

			restarted using the current values of the test parameters.	
Interface	string(256)	W	IP-layer interface over which the test is to be performed. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE MUST use the default routing interface.	-
UploadURL	string(256)	W	The URL as defined in [3], for the CPE to Upload to. This parameter MUST be in the form of a valid HTTP [2] or FTP [6] URL. When using FTP transport, FTP binary transfer MUST be used. When using HTTP transport, persistent connections MUST be used and pipelining MUST NOT be used. When using HTTP transport the HTTP Authentication MUST NOT be used.	-
DSCP	unsignedInt[0:63]	W	DiffServ code point for marking packets transmitted in the test. The default value SHOULD be zero.	-
EthernetPriority	unsignedInt[0:7]	W	Ethernet priority code for marking packets transmitted in the test (if applicable). The default value SHOULD be zero.	-
TestFileLength	unsignedInt	W	The size of the file (in bytes) to be uploaded to the server. The CPE MUST insure the appropriate number of bytes are sent.	-
ROMTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the client sends the PUT command For FTP this is the time at which the STOR command is sent.	-
BOMTime	dateTime	-	Begin of transmission time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the first data packet is sent. For FTP this is the time at which the client receives the ready for transfer notification.	-
EOMTime	dateTime	-	End of transmission in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456	-

			For HTTP this is the time when the HTTP successful response code is received. For FTP this is the time when the client receives a transfer complete.	
TotalBytesSent	unsignedInt	-	The total number of bytes sent on the Interface between BOMTime and EOMTime.	-
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the time at which the TCP socket open (SYN) was sent for the HTTP connection. For FTP this is the time at which the TCP socket open (SYN) was sent for the data connection Note: Interval of 1 microsecond SHOULD be supported.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For HTTP this is the Time at which the TCP ACK to the socket opening the HTTP connection was received. For FTP this is the Time at which the TCP ACK to the socket opening the Data connection was received. Note: Interval of 1 microsecond SHOULD be supported.	-
ConcurrentSessions	unsignedInt	-	The number of concurrent connections to use in a multi-threaded test	-
TestBytesSentUnderFullLoading	unsignedInt	-	Test traffic sent by the CPE between the last BOMTime and first EOMTime across all connections in a multi-threaded test.	-
TotalBytesSentUnderFullLoading	unsignedInt	-	The total traffic sent in bytes between the last BOMTime and first EOMTime across all connections in a multi-threaded test.	-
TimeBasedTestIncrements	unsignedInt	-	When TimeBasedTestIncrements > 0, HTTP/FTP/NDT transactions are time segmented into subintervals (i.e. time slots) , each having a duration of TimeBasedTestIncrements (seconds). Defaults to zero, which implies time segmenting is disabled. TimeBasedTestIncrements is in units of seconds.	-
TimeBasedTestIncrementsOffset	unsignedInt	-	Normally TimeBasedTestIncrements segments HTTP/FTP/NDT transactions into subintervals of duration TimeBasedTestIncrements starting from BOMTime (or the last BOMTime in a multi-threaded transaction). However, when TimeBasedTestIncrementsOffset > 0, the TimeBasedTestIncrements start at time BOMTime (or the last BOMTime in a multi-threaded transaction) +	-

			TimeBasedTestIncrementsOffset. It is used to remove the contribution from slow start in a throughput measurement. TimeBasedTestIncrementsOffset is in units of seconds.	
.UploadDiagnostics. .IncrementalResults{i}	object	-	Results for time segmented throughput tests. A new object is created every TimeBasedTestIncrements	-
IncTestBytesSent	unsignedInt	-	Incremental change in the value of TestBytesSentUnderFullLoading every TimeBasedTestIncrements seconds between the last BOMTime and first EOMTime for a multi-threaded test	-
IncBOMTime	dateTime	-	Empirical measurement of BOMTime every TimeBasedTestIncrements seconds	-
IncEOMTime	dateTime		Empirical measurement of EOMTime every TimeBasedTestIncrements seconds	-

Table 3 (TR-143 Test Control Variables – Phase 2)

[3.2.2] When throughput testing is enabled (ThroughputTestingEnabled = True), the client device (e.g. BHR) must send a TestRequest packet to the test server due to one of three events:

1. the date/time indicated in NextRequestTime field of the previous TestResponse packet which had NRT = 1 was reached, or
2. the time epoch corresponding to a device scheduled time in the current TestRequestInterval (as specified in [3.1.2]) is reached and NRT in the previous TestResponse packet was 0 (i.e. the TestRequestInterval expired), or
3. DiagnosticState is set to "Request" at which point a TestRequest packet is immediately sent to the test server.

Each Home Router (BHR) must be assigned a unique random seed for device scheduled (uniformly distributed) random time selection. That is, by specifying a unique seed for the each BHR the TestRequest's emanating from the BHRs can be randomized.

Note that DiagnosticState = "Request" is used for on-demand testing with test control, which DiagnosticState = "Requested" is used for on-demand testing without test control (i.e. DiagnosticState = "Requested" does not send a TestRequest packet and perform the test according to a TestResponse packet as it directly performs the test configured in the device).Also note that a DiagnosticState = "Request" (i.e. test controlled on-demand test) or DiagnosticState = "Requested" (on demand test without test control) triggered test occurs independently to the device scheduled test (whether from a device scheduled per TestRequestInterval or for a NRT = 1 in the previous TestResponse packet). However, if an on-demand test were to occur simultaneously with a device scheduled test, the on-demand test would have priority and can preempt a device scheduled test. However,

the device scheduled test would resume during the next cycle (i.e. during the next TestRequestInterval).

For the device scheduled tests, when ThroughputTestingEnabled = True, TestRequest's are recurring with one attempt every TestRequestInterval when NRT is False (0) in the previous TestRequest message. If NRT was True (1) in the previous TestRequest message, then the NextRequestTime field of that message is explicitly used to determine the next time to send a test request packet as long as NextRequestTime is within two TestRequestInterval's from the current time (i.e. is within current time + 2*TestRequestInterval)[2].

- By current time, we mean the time the TestResponse is received.
- The NextRequestTime is the UTC date/time in dateTime string format indicating when the client (BHR) will make its next TestRequest.
- If the value of NextRequestTime is not within the current time + 2*TestReqestInterval, then it is ignored and the device scheduled request time for the next TestRequestInterval is used (as specified in [3.1.2]).

To inform the test control server that a given TestRequest packet is sent at a time epoch due to a NextRequestTime of the previous TestResponse packet (i.e. NRT was set to 1 in that TestResponse packet), the client should set NRT = 1 in the current TestRequest packet. However, if a TestRequest packet is sent at a time epoch corresponding to a device scheduled random time in the current TestRequestInterval (as specified in [3.1.2]), the client should set NRT = 0 in the TestRequest packet.

So in summary, the command to use NextRequestTime to determine the time to send the next TestRequest packet is dictated by the test control server setting the NRT bit to NRT = 1 in a TestResponse packet. When NRT is set to 1 by the server, then it must also place the UTC time for the client (BHR) to make it's next TestRequest in the NextRequestTime field (using dateTime string format). If NextRequestTime is determined to be valid by the client (i.e. is within two TestRequestInterval's from the time the TestResponse is received) then the client sends the next TestRequest packet at NextRequestTime explicitly, and sets NRT = 1 in that TestRequest packet. Otherwise it sends the next TestRequest at the device scheduled time in the next TestRequestInterval and sets NRT = 0.

The TestRequest packets are sent to the UDP Port TestRequestUDPPort of the test server which has IP address TestServerIPAddress. The TestRequest packet is a UDP packet having the following payload format:

2

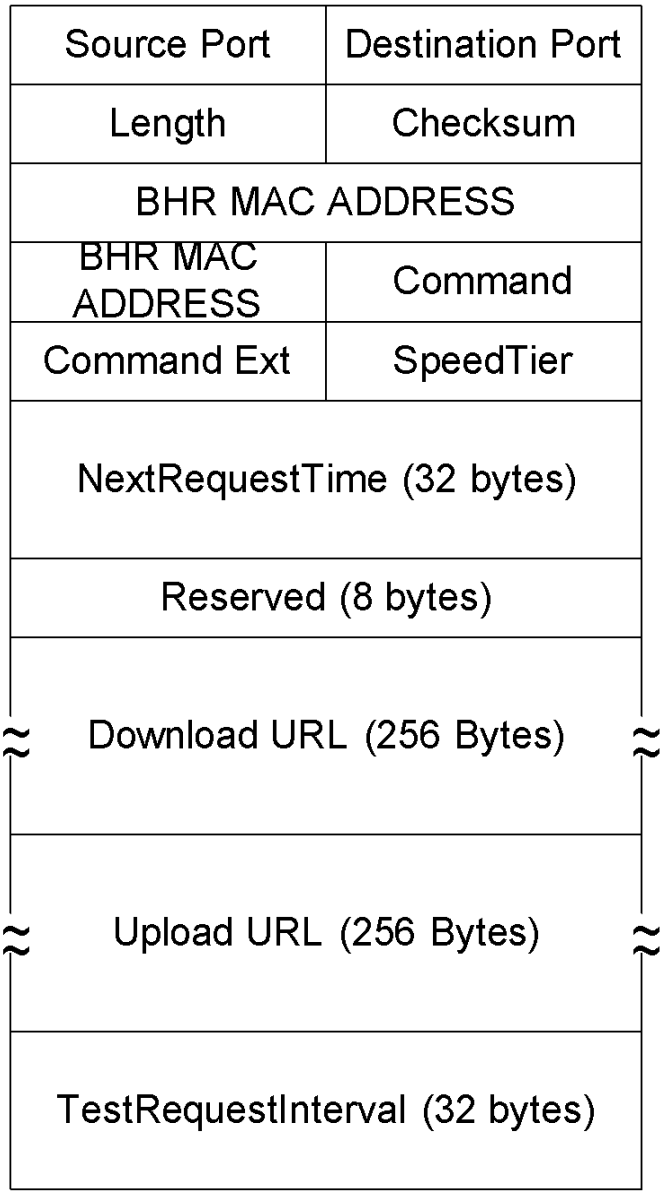

Figure 6 (TestRequest packet format)

The TestRequest packet has a payload of 596 bytes, eight of which are "Reserved" for future inclusion of other parameters. For example, more provisioning information regarding the subscriber such as his/her OLT CLLI, GWR CLLI, etc …can be provided if known. The BHR MAC address is a 6-byte binary encoding of the MAC address for the interface of the client device the test is run on. *As a point of interest, note that the IP address to MAC address mappings afforded by TestResquest packets may be used by the test server to determine which IP's to perform UDP Echo Plus tests to.*

The format of the 16 bit “Command” field is:

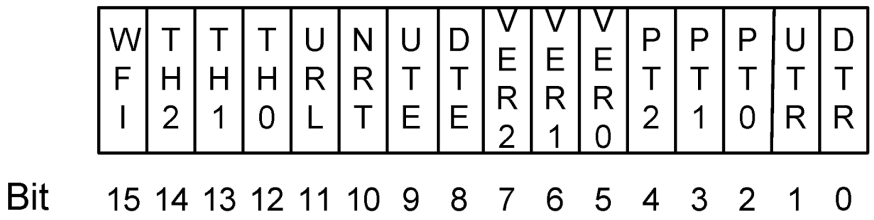

Figure 7 (Format for Command Field)

The least significant bit of the “Command” field is the DTR (Download Test Request) bit and the second bit is the UTR (Upload Test Request). DTR is set to 1 if PerformDownloadThroughputTest = True, and UTR is set to 1 if PerformUploadThroughputTest = True for every TestRequest packet sent by the BHR. Note that TestRequest packets are not sent at all if ThroughputTestingEnabled = False. The test control server, upon receiving the TestRequest packet will either return the packet (now technically referred to as a TestResponse packet) to the BHR with the DTE (Download Test Enabled) bit set or unset, the UTE (Upload Test Enabled) bit set or unset, or not return the TestRequest packet at all (e.g. if it is too busy).
After sending the TestRequest packet, the BHR will wait for TestRequestTimeout (milliseconds) for a reply (i.e. a TestResponse packet being returned) from the test server. If it is not received within that time interval, the test is not performed for that cycle and the TestRequest packet is sent again at a random time in the next TestRequestInterval. If the TestResponse packet is returned from the server, a download test is performed if DTE is set to 1, and an upload test is performed if UTE is set to 1. Both DTE and UTE are set to zero when the TestRequest packet is first sent to the server. *The same protocol must be used for both a downstream and upstream test if both are performed.* If the 3 bit Protocol type (PT) field is set to all zeroes (PT2 = 0, PT1 = 0, PT0 = 0) in the TestResponse packet, then the HTTP protocol is used for the download and/or upload test(s). If the PT field is set to one (PT2 = 0, PT1 = 0, PT0 = 1), then the FTP protocol is used for the download and/or upload throughput test(s). If the PT field is set to 2 (PT2 = 0, PT1 = 1, PT0 = 0) then the Network Diagnostic Tool (NDT) protocol is used for the download and upload throughput test(s). If the PT field is set to 3 (PT2 = 0, PT1 = 1, PT0 = 1), then the DNS response time test is performed and if the PT field is set to 4 (PT2 = 1, PT1 = 0, PT0 = 0) then a Web Page response time test is performed. If the PT field is set to 5 (PT2 = 1, PT1 = 0, PT0 = 1) then a Reverse UDP Echo Plus test (where “Reverse” is used to denote that the BHR functions as a UDP Echo client) is performed on the UDP port specified by the UDPPort parameter in Table 2. If the PT field is set to 6 (PT2 = 1, PT1 = 1, PT0 = 0), then the TWAMP test is performed. If the PT field is set to 7 (PT2 = 1, PT1 = 1, PT0 = 1) and the CommandExt field in Figure 6 (TestRequest packet format) is set to 0x00 then a PathPing test is performed and if the PT field is set to 7 (PT2 = 1, PT1 = 1, PT0 = 1) and the CommandExt field in Figure 6 (TestRequest packet format) is set to

0x01 (decimal 1) then a PassiveUsageMonitoring test is performed at the BHR. If the PT field is set to 7 (PT2 = 1, PT1 = 1, PT0 = 1) and the CommandExt field in Figure 6 (TestRequest packet format) is set to 0x02 (decinal 2) then a UDPThroughput test (as defined in Appendix I) is performed.

Note that if an NDT test (PT field set to 2), a DNS response time (PT field set to 3), Web Page response time (PT field set to 4) a Reverse UDP Echo Plus test (PT field set to 5), a TWAMP test (PT field set to 6) test, a PathPing test (PT field set to 7 and CommandExt field set to 0x00), or a PassiveUsageMonitoring test (PT field set to 7 and CommandExt field set to 0x01) is requested, then only the Download Test Enable (DTE) bit in the TestResponse applies. That is, the UTE bit is ignored for a DNS response time, Web Page response time, Reverse UDP Echo Plus tests, TWAMP test, PathPing test, or PassiveUsageMonitoring test, since these tests occur in one direction. This condition also applies to the NDT test because the NDT will always be performed bi-directionally.

The Protocol type bits are initially set by the BHR according to the RequestedTestProtocol parameter as specified in Table 3. For example, an HTTP test is specified by RequestedTestProtocol = "HTTP" and a valid HTTP URL of the form http://server_ip_address (or http://fileserver.com) set in DownloadURL and UploadURL, or an FTP test specified by RequestedTestProtocol = "FTP" and a valid FTP URL of the form ftp://server_ip_address (or ftp://fileserver.com) set in DownloadURL and UploadURL, etc… Custom URL formats are devised for the remaining protocols and are described later in this document. Note that both DownloadURL and UploadURL should correspond to RequestedTestProtocol and hence they should specify the same test protocol since only a single test protocol can be used for download and upload components of a bi-directional throughput test. If either DownloadURL or UploadURL suggests a differenet test protocol then the RequestedTestProtocol (e.g. RequestedTestProtocol = "HTTP" while DownloadURL = ftp://server_ip_address/download_file.txt), then the corresponding DTR or UTR bit is set to 0 in the TestRequest packet.

Note that, just as in the case of the HTTP, FTP, and NDT test, the test server can dictate to which IP address a DNS test or RevereUDPEchoPlus test or TWAMP test or PathPing test is performed by setting a target IP address (or hostname) in the URL field of the TestResponse packet. All of the supported test protocol label options ("HTTP, "FTP", "NDT", "DNS", "WebPage", "ReverseUDPEchoPlus", "TWAMP", "PathPing", "PassiveUsageMonitoring", and "UDPThroughput") should be included in the DownloadTransports, and all of the protocols that support operation in the upstream direction ("HTTP", "FTP", and "UDPThroughput") should be included in the UploadTransports parameters specified in Table 1. That is, DownloadTransports and UploadTransports should each contain a comma separated list of all the test protocols supported by the BHR. Currently "HTTP" and "FTP" are the only test protocol options specified in the TR143 standard and we are pushing to have these additional test protocol options ("NDT", "DNS", "WebPage", "ReverseUDPEchoPlus", "TWAMP", "PathPing", "PassiveUsageMonitoring", and "UDPThroughput") incorporated in a later revision of

the TR-143 standard. *Note that for Phase 2 it is possible for the test server to change the test protocol used for performing a throughput or response time test (from the protocol requested) to a different protocol in the TestResponse packet. The PT bit settings in the "TestResponse" packet dictate the actual protocol that is used to perform the test. This provides the test server the ability to modify the test protocol or destination used for the test.* The protocol value provided in RequestedTestProtocol serves as the default test protocol since this is the protocol that is requested in the TestRequest packets but not necessarily the protocol that will be used to perform the test. As previously mentioned DownloadURL and UploadURL should be set to the same protocol type (e.g. HTTP or FTP or NDT, etc …) when the Test Request/Response control protocol is used to provide periodic scheduling given the single test type protocol limitation of the TestRequest packet. If they are set to different values, then only the DownloadURL or UploadURL value that match the test protocol set in the ProtocolType (PT) bits is applicable for that test. For example, if the test protocol (i.e. PT bits) in TestRequest message is HTTP and DownloadURL is http://testsever/file.txt while UploadURL is ftp://testserver/file1.txt, then the HTTP protocol is assumed to be requested for the download direction only. If on the other hand, the test protocol (i.e. PT bits) in TestRequest message is HTTP and DownloadURL is ftp://testsever/file.txt while UploadURL is http://testserver/file1.txt, then the HTTP protocol is assumed to be requested for the upload direction only.

The version (VER) field (bits 5, 6, and 7), is a 3-bit field that indicates the version of the client that is making the TestRequest. The 3-bits provide for a distinction of 8 different versions. These bits are set to a hexadecimal coding of the ClientVersionNumber variable in Table 3. That is, when ClientVersionNumber = 0 then VER2 = 0, VER1 = 0, VER0 = 0, or when ClientVersionNumber = 2 then VER2 = 0, VER1 = 1, VER0 = 0, etc … Since Phase 1 clients will not set these bits, a coding of zero (VER2=0, VER1=0, VER0=0) denotes a Phase 1 client and hence the default value of ClientVersionNumber for a Phase 1 client is ClientVersionNumber = 0. Similarly, the default value of ClientVersionNumber for a Phase 2 client is ClientVersionNumber = 1 (resulting in a coding of VER2 = 0, VER1 = 0, VER0 = 1) and so on. This provides the test control server with information as to whether the client will interpret and respond to Phase 2 command field settings.

The NextRequestTime (NRT) bit (bit 10) is set by the test server in the TestResponse packet and is used to indicate an explicit next time for sending the next TestRequest packet. When NRT = 1, then the (32 byte) NextRequestTime field provides a string value for the next time for the BHR to send a TestRequest. That is, when NRT = 1, then the NextRequestTime field contains a NULL terminated string of the form "2008-04-09T15:01:05.123456" which denotes the prospective NextRequestTime (i.e. time to send the next TestRequest packet) to the microsecond (in UTC). If microseconds resolution is not supported on the BHR, then the value is interpreted to millisecond resolution or to whatever resolution the BHR does support. However, when a NextRequestTime value is provided, the BHR must first determine if that value falls within two TestRequestInterval's from the time when the TestResponse packet (that provided the NextRequestTime value) was received. If it does not, then it is ignored and

the device scheduled random time within the current TestRequestInterval is used to trigger the next TestRequest as normally is the case (per requirement [3.1.2]). The purpose of including the explicit NextRequestTime feature is to enable the test server to better balance its test request load over time. The NRT bit is set to 1 by the client in a TestRequest packet that is being sent as a result of an NRT value provided in the previous TestResponse packet (i.e. at the time specified by the NextRequestTime field in the previous TestResponse packet it received). That is, the BHR sets the NRT bit to 1 in a TestRequest packet that is sent as a consequence of a previous TestResponse packet (having a valid NextRequestTime field setting) with NRT set to 1. If NextRequestTime was not provided or enabled in the previous TestResponse packet (e.g. if NRT was zero in the previous TestResponse packet), then the client is sending at a device scheduled random time per its own initiative and in that case sets the NRT bit equal to 0. Note that a client that supports the NextRequestTime capability described here should also set the TestRequestInterval field in the TestRequest packet to an string representation of the TestRequestInterval parameter (in seconds) as defined in Table 3.

The URL (URL) bit (bit 11) is used to re-direct the BHR to a particular resource to download from and/or upload to. When URL = 1, then the NULL terminated strings starting at byte 52 (Download URL field) and byte 308 (Upload URL field) in the TestResponse packet are used instead of the DownloadURL and UploadURL variables in the BHR as specified in Table 3. Note that the Upload URL string is just a destination for sending the dummy file generated by the BHR upstream. When URL = 1, then both the Download URL field and Upload URL field values are provided so the single URL bit is used for both test directions. When URL=1 and an invalid Download URL is received by the client, then the DownloadURL parameter defined in Table 3 is used instead. Similarly, when URL=1 and an invalid Upload URL is received by the client, then the UploadURL parameter value defined in Table 3 is used instead. Part of validating a URL is confirming the UploadURL field and/or DownloadURL field matches the test protocol type (e.g. ftp://ftp.fileserver.com is a valid URL for an FTP protocol test (PT2 = 0, PT1 = 0, PT0 = 1) but not valid for and HTTP protocol test (PT2 = 0, PT1 = 0, PT0 = 0)). When the NDT test protocol is used (i.e. when PT2 = 0, PT1 = 1, PT0 = 0), then the Download URL field contains a string of the form:
“ndt://NDTServerHostname:NDTServerTCPport” or
“ndt://NDTServerIPaddress:NDTServerTCPport” (e.g. ndt://10.20.42.34:3001 indicates an NDT test is run to an NDT server at IP address 10.20.42.34 over NDT control port 3001) . The NDT test also supports options for multiple threads (using the TH bits) and user selectable test duration, time based test increments, and time based test increments offset using additional colon delimited variables in that order. That is, the NDT URL can be specified in the followig format,
“ndt://NDTServerHostname:NDTServerTCPPort:TestDuration:TestIncrements:TestIncrementsOffset”, when an NDT test is run to a compliant NDT server. The definition of TestIncrements and TestIncrementOffset are provided in Appendix A – Extended Capabilities for the HTTP Test, as they are identical to the definitions for the HTTP throughput test. For example, the URL
“ndt://NDTServerHostname:NDTServerTCPport:30:5:1” in the TestRequest packet

means the BHRis requesting to run an NDT test (using the number of threads specified by the TH bits as described below) having a test duration of 30 seconds, with test increments (i.e. throughput snapshots) to be taken every 5 seconds, starting from an offset of 1 sec. Note that when the test duration or test increment or test increment offset variables are not provided in the URL, the default values for these paramters (assumed by the test server) are test duration = 10 sec, test increments = 0 (i.e. disabled), and test increment offset = 0 (i.e. disabled). Similarly, if NDTServerTcpPort is not provided in the URL, then the default value assumed by the test server is 3001. Note that, regardless of what value are requested in the TestRequest packet, the actual test run is specified by the values (PT bits, TH bits, DownstreamURL and UpstreamURL) in the TestResponse packet so the requested parameters may be modified by the test server. The UploadURL field is not applicable for the NDT test since each NDT must be bi-directional. Since some of the test types (e.g. NDT and DNS response time) do not have standard URLs, in the following table we provide the URL formats that will be used for each test type:

Test Protocol	**URL Format**	**Support s Multi-Thr eaded**	**Supports Downstrea m URL**	**Supports Upstream URL**
HTTP	http://ServerHostna me:port	Yes	Yes	Yes
FTP	ftp://ServerHostnam e:port	No	Yes	Yes
NDT	ndt://ServerHostnam e:port	Yes	Yes	No[3]
DNS	HostName	No	Yes	No
Web Page	http://ServerHostna me:port	No[4]	Yes	No
Reverse UDP Echo Plus	HostTarget (or HostTarget:port)	No	Yes	No
TWAMP	twamp://TwampCon trolServerHostname: port (port is only supplied if the well defined control port of 862 for Twamp is to be overidden)	No	Yes	No
PathPing	PathPingHostTarget	No	Yes	No

[3] The NDT supports both and upload and download test natively and thus does not require an explicit upload test be specified in the TestResponse packets.

[4] Web Page download tests are natively multi-threaded per the HTTP 1.1 (or HTTP 1.0) protocol and thus the numbers of threads do not need to be specified in the TestResponse packets.

	(or PathPingHostTarget:port)			
PassiveUsageMonitoring	Not Applicable	No	No	No
UDPThroughput	UdpThroughput://ServerHostname:TestDuration:TestIncrements:port1:port2:port3:port4	Yes	Yes	Yes

Table 4

The client behaviors for the HTTP and FTP test types are defined in the TR-143 standard. However, we extend the HTTP test protocol to support multiple connections (multi-threaded), time-based test duration, and sampling of bytes counts at fixed time increments (i.e. test increments). The client behaviors for this and the remaining test types are defined in the appendixes of this document.

The number of test threads (TH) trio of bits TH2, TH1, TH0 (bits 14, 13, and 12), indicate the number of TCP connections to use for a multi-threaded HTTP or NDT download or upload test. That is, TH2=0, TH1=1, TH0=1, indicates that three concurrent connections should be used to perform the test and so forth. The TH bits only apply to multi-threaded throughput test protocols (e.g. HTTP and NDT) and are ignored when all the bits are zero. Therefore, there is no difference between the TH bits being set to binary zero (i.e. TH2=0, TH1=0, TH0=0) and the TH bits being set to binary 1 (i.e. TH2=0, TH1=0, TH0=1) since both imply a single thread is used. *Currently only HTTP and NDT are supported for multi-threaded testing.*

The Wait for Idle (WFI) bit (bit 15) denotes the BHR should offset the time it sends its next TestRequest packet (whether it is due to a device scheduled time selection in the next TestRequestInterval or the explicit time provided by the NextRequestTime field in the TestResponse packet) until after LowActivityWaitTime seconds have passed with the WAN port utilization having a value less than LowActivityThreshold (bits/sec) as in requirement [3.1.3]. Thus WFI = 1 is just an explicit indication to perform requirement [3.1.3] (i.e. WFI bit serves as an On/OFF switch for the feature) specifically for the next TestRequest packet to be sent.

The CommandExt field is a 16 bit field used to extend the options (i.e. number of bits) for particular fields in the Command field. Currently only two values are used in the CommandExt field and they apply when the PT bits setting in the Command field is 7 (i.e. PT2 = 1, PT1 = 1, PT0 = 1). When the PT bits are set to 7, then a CommandExt field value of 0x00 (i.e. all zeroes) means that the test protocol is the PathPing test. Also, when the PT bits are set to 7, then a CommandExt field value of 0x01 (i.e. all zeroes except the least significant bit of the CommandExt field set to 1) means that the test protocol is the

PassiveUsageMonitoring test. Thus currently the CommandExt field is used to extend the PT bits subfield in the Command field.

The SpeedTier field is a 16 bit field used to communicate the speedtier settings in the BHR to the test server. This field supports the test admission control function in the test server. The speedtier settings are defined by the values assigned to the DownstreamSpeedTier and UpstreamSpeedTier parameters in InternetGatewayDevice.X_ACTIONTEC_PerformanceDiagnostics. The most significant byte of the speedtier field is used to denote a quantized value for DownstreamSpeedTier and the least significant byte is used to denote a quanted value for UpstreamSpeedTier. Hence we refer to the most significant byte and least significant byte subfields as the DownstreamSpeedTier subfield and UpstreamSpeedTier subfield, respectively. Both the DownstreamSpeedTier subfield and UpstreamSpeedTier subfield are coded as unsigned integer values in the TestRequest packet even though they are derived from string values placed in the DownstreamSpeedTier and UpstreamSpeedTier parameters of Table 3 (TR-143 Test Control Variables – Phase 2), after converting those parameters to unsigned integers and dividing by the SpeedTierScaling parameter defined in Table 3 (TR-143 Test Control Variables – Phase 2). For example, if we have DownstreamSpeedTier = “25Mbps” and UpstreamSpeedTier = “5Mbps” with SpeedTierScaling = 5, as defined in Table 3 (TR-143 Test Control Variables – Phase 2), the BHR would set the SpeedTier field as follows. First the values of DownstreamSpeedTier and UpstreamSpeedTier are converted to unsigned integers ignoring the units so we have a downstream speedtier value of 25 and an upstream speedtier value of 5. Then after dividing each of these unsigned integer representations by the SpeedTierScaling, we have values of 5 (for downstream) and 1 (for upstream) so the value of the DownstreamSpeedTier subfield is coded as 5 hex and the value of the UpstreamSpeedTier subfield is coded as 1 hex. Since each SpeedTier subfield is 1 byte, we can only send one of 256 values in each subfield so the SpeedTierScaling value provides greater range in speedtier values that can be conveyed. If SpeedTierScaling does not divide evenly into a downstream or upstream speedtier value then the Ceiling function is used to provide the upper integer value. So a DownstreamSpeedTier = “25Mbps” and UpstreamSpeedTier = “.768Mbps” with SpeedTierScaling = 4 would result in the values 7 (for downstream) and 1 (for upstream) being placed in the DownstreamSpeedTier subfield and UpstreamSpeedTier subfield, respectively. Also if the DownstreamSpeedTier or UpstreamSpeedTier or SpeedTierScaling parameters are unassigned or have illegitimate values, then a value of zero is placed in each respective subfield of the SpeedTier field. Also, if the division results in a value greater than 255, then the value of 0 is placed in the respective subfield (i.e. the the max value for each subfield of the SpeedTier field is 255).

[3.2.3] same as [3.1.3] with the additional requirement that if LowActivityWaitExpirationTime expires before a TestRequest packet can be sent that complies with requirement [3.1.3] (that is, LowActivityWaitTime seconds have passed with the WAN port utilization having a value less than LowActivityThreshold (bits/sec)), then the TestRequest is not sent for the “current” TestRequestInterval. If a

TestRequestInterval boundary is crossed while waiting for requirement [3.1.3] to be met then the latter TestRequestInterval is considered to be the "current" TestRequestInterval. The purpose of this requirement is to trigger tests during periods of low to no subscriber activity.

[3.2.4] same as [3.1.4]

[3.2.5] same as [3.1.5]

[3.2.6] is the same as [3.1.6] with the following additional requirements. Since multiple test protocols are supported for Phase 2 clients, we need to specify some additional test result messages beyond the TestResult message defined in Figure 4 (which is specified for a standard HTTP or FTP throughput test). For some tests, several messages will be required to pass the results back to the test server. For all test types we specify that the final message sent back to the server is a TestLocation message. For an HTTP or FTP test, when TimeBasedTestIncrements = 0, TestLocation packet should be sent following the one or two TestResults messages on the same TCP connection opened for the TestResult message(s) (which has a TCP port number equal to TestResultsTCPPort). Recall that two TestResult's messages are sent sequentially to provide the results for both an upstream and downstream tests that were performed back to back, while a single TestResult message is used to provide the results for a single direction test (upstream or downstream). When TimeBasedTestIncrements > 0, then TestIncrement messages are sent following each TestResult message. Following the one or two TestResult's messages and associated TestIncrement messages sent to the test control server, the TestLocation message is also sent before closing the connection. The TestLocation message is used to provide some additional information about the ID and location of the BHR that just performed the test(s) to the test control server. The TestLocation message has the following packet format:

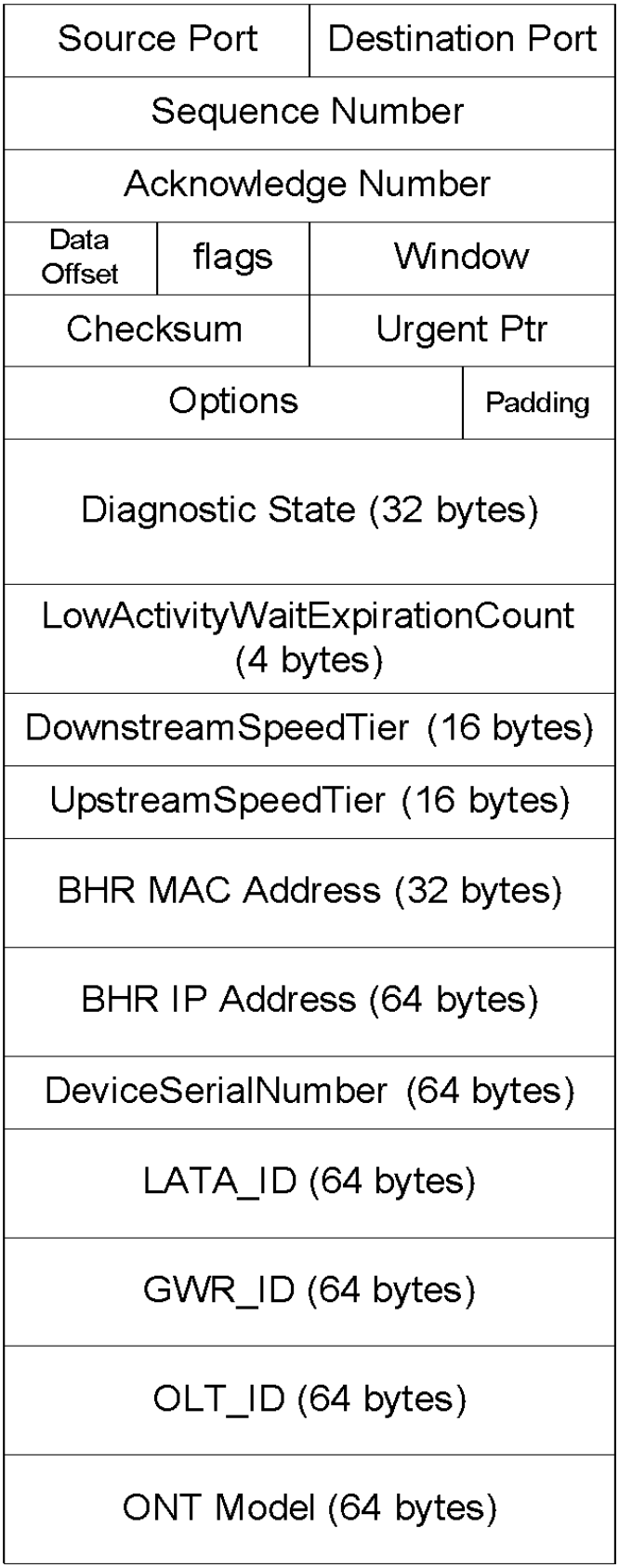
Source Port
Destination Port
Sequence Number
Acknowledge Number
Data Offset
flags
Window
Checksum
Urgent Ptr
Options
Padding
Diagnostic State (32 bytes)
LowActivityWaitExpirationCount (4 bytes)
DownstreamSpeedTier (16 bytes)
UpstreamSpeedTier (16 bytes)
BHR MAC Address (32 bytes)
BHR IP Address (64 bytes)
DeviceSerialNumber (64 bytes)
LATA_ID (64 bytes)
GWR_ID (64 bytes)
OLT_ID (64 bytes)
ONT Model (64 bytes)

Figure 8 (TestLocation packet format)

The first (32 byte) field is the "Diagnostic State" field. Note that this field is also included (at the same position) in a TestResult message and thus its value is used to distinguish TestResult messages from TestLocation messages. For a TestLocation message the Diagnostic State field must be set to the string value "TestLocation". *Note: For a TestResult message the Diagnostic State field will have one of a finite set of string values denoting the completion state of the last test performed (and thus the string "TestLocation" is distinct).*

The 4 byte LowActivityWaitExpirationCount field is encoded in binary, but all the remaining fields in the TestLocation packet are encoded as character strings. Verizon will use a LATA_ID (64 bytes), GWR_ID (64 bytes), OLT_ID (64 bytes), and ONT_ID (64 bytes) strings to identify the location of the BHR, and the UpstreamSpeedTier (16 bytes) and DownstreamSpeedTier (16 bytes) strings to identify the speed tier of the BHR that sent the TestLocation packet (as a result of just completing a throughput test). Note, the ONT_ID (64 byte) field could be used to denote the ONT model type that is serving the BHR to delineate performance measures by different ONT types. The BHR itself is identified by its (64 byte) serial number (Device Serial Number), its IP address (BHR IP Address), and MAC address (BHR MAC Address). Note that the BHR IP Address field is 64 bytes as it must account for IPv6 addresses. The BHR MAC Address field is 32 bytes (as opposed to 6 bytes in the TestRequest packet) since it is encoded as a character string in the TestLocation packet (in contrast to the binary encoding for the BHR MAC Address in the TestRequest packet).

[3.2.6.1] A Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection can be used in in place of the TCP connection for sending the TestResult(s) and TestLocation messages.It is required that the device support OpenSSL as an option for transporting the test results.

[3.2.7] same as [3.1.7] with the addition that UDPEchoPlus be able to function simultaneously with "any" of the other remaining test types (HTTP throughput tests, FTP Throughput tests, NDT Throughput test, DNS ReponseTime test,Web Page ResponseTime test, ReverseUDPEchoPlus test, TWAMP test, PathPing test, or PassiveUsageMonitoring test). Note that that UDPEcho (or UDPEchoPlus) reflector function is not considered to be a test type and should operate independently of all of the remaining test types supported in Table 4.

[3.2.8] same as [3.1.8] with the additional "optional" requirement that the NDT throughput test be capable of supporting the time based increments feature in addition to multi-threaded throughput testing specified in [3.1.8]. That is, it should be capable of taking the time-based snapshots every TimeBasedIncrement starting from an offset of TimeBasedIncrementOffset as specified in Appendix A for the generalized HTTP throughput test.

[3.2.9] Add a page to the user web interface that a user can use to configure the throughput test variables (as specified in TR-143 and this document) and to either enable a single throughput test (by setting DiagnosticState to “Requested”) and/or enable periodic/scheduled testing as specified in this document. Provide a mechanism for setting all relevant test variables and reading all the relevant test result variables. Also provide access to the variables required to enable the UDP Echo Plus responder function.

Appendix A – Extended Capabilities for the HTTP Test

This section provides specifications that extend the capabilities of the HTTP test protocol to support a multi-threaded mode (Multi-Threaded HTTP), a time-based test duration mode (Time Based HTTP), and incremental sampling of received or sent bytes counts. These additional capabilities (multi-threaded mode and time based mode, and byte count sampling) are not mutually exclusive. That is, an HTTP test can be both multi-threaded and time-based, and in either case have incremental sampling enabled or disabled. By multi-threaded mode we mean the test supports multiple simultaneous TCP sessions (connections) for performing the test. Time based mode means the download/upload test runs for a pre-determined period of time (e.g. 30 seconds) as opposed to being based on file size (i.e. transferring a particular file).

The multi-threaded mode is based on BBF contribution bbf2010.420.01; hence that contribution may be referenced for providing some additional discussion on the multi-threaded mode. Essentially bbf2010.420.01 extends the HTTP test to support multiple concurrent sessions for the purpose of reducing the required TCP window sizes for each session. Multi-threaded throughput tests have a distinct response to packet loss compared to a single threaded test because multiple sessions recover from a packet loss simultaneously resulting in different convergence of the congestion control and avoidance of the aggregate flow. Each session in a multi-threaded download attempts to download the same file specified by DownloadURL, but the number of test bytes measured “while all sessions are active” is referred to as the TestBytesReceivedUnderFullLoading. Similarly, each session in a multi-threaded upload attempts to write TestFileLength bytes to the location specified by UploadURL but the number of test bytes sent “while all sessions are active” is called TestBytesSentUnderFullLoading. Note that TestBytesReceivedUnderFullLoading and TestBytesSentUnderFullLoading are the number of bytes received or sent starting from when the last session is opened until the first session is closed as that span represents the time that all sessions are active. The multi-threaded test mode is used when the parameter ConcurrentSessions in DownloadDiagnostics and/or UploadDagnostics is set to a value greater than 1. ConcurrentSessions has a maximum value of 7 (indicating 7 concurrent

sessions are to be used) and multi-threaded testing is not in effect (i.e. single sessions is used as in the standard HTTP test) when ConcurrentSessions is 0 or 1.

Time based test mode is specified for a download when a download filename having the form "timebasedmode_xx.txt", where xx represents an unsigned number of seconds (e.g. 30), is requested from the test server. Thus, a DownloadURL of http://ServerHostname/timebasedmode_30.txt specifies that the server supply data so that the download takes exactly 30 seconds to complete. Similarly, an UploadURL of http://ServerHostname/timebasedmode_30.txt specifies that the BHR forward upstream data (starting from when all sessions are active for a multi-threaded test) for 30 seconds and close the connection (or close the first connection in multi-threaded test) at the 30 second time epoch. TR-143 test servers will interpret these special URLs in the same manner.

Another measurement feature that can be enabled is the test increments sampling function. This feature can be enabled independent of whether or not the HTTP test is single threaded, multi-threaded, or time based, by setting the TimeBasedTestIncrements (unsignedInt - seconds) parameter to a value greater than zero (otherwise it is disabled). When TimeBasedTestIncrements is greater than zero, then every TimeBasedTestIncrements seconds, a timestamp is made and set to DownloadDiagnostics.IncrementalResults{i}.EOMTime and the current value of TestBytesReceivedUnderFullLoading is set to DownloadDiagnostics.IncrementalResults{i}.IncTestBytesReceived for a download. Similarly, when TimeBasedTestIncrements is greater than zero, then every TimeBasedTestIncrements seconds, a timestamp is made and set to UploadDiagnostics.IncrementalResults{i}.EOMTime and the current value of TestBytesSentUnderFullLoading is set to UploadDiagnostics.IncrementalResults{i}}.IncTestBytesSent. Note that if TimeBasedTestIncrements does not divide evenly into the length of time required to complete the download or upload (whether it be time-based or a file transfer based), then the actual EOMTime of the download or upload (or the EOMTime for the first session to complete in a multi-threaded test) is used as the last timestamp (i.e. DownloadDiagnostics.IncrementalResults{i}.EOMTime or UploadDiagnostics.IncrementalResults{i}.EOMTime) value. Hence the last time interval may be less than TimeBasedTestIncrements seconds. Also, note that for multi-threaded tests, the IncTestBytesReceived and IncTestBytesSent are measured across all the concurrent sessions. Hence they begin the first data unit after the last session is opened and end on the last data unit when the first session is closed. This is why we use TestBytesReceivedUnderFullLoading and TestBytesSentUnderFullLoading. Note that for a single thread test, TestBytesReceivedUnderFullLoading and TestBytesReceived are identical, and TestBytesSentUnderFullLoading and TestFileLength are identical. But for a multi-threaded test TestBytesReceived and TestFileLength are measured from the first data unit following the "first" BOMTime to last data unit just prior to the "last" EOMTime.

[A.1] The device must support all of the variables introduced in Table 3 which include the standard HTTP throughput test parameters plus the following additional variables (in DownloadDiagnostics and UploadDiagnostics) to support multi-threaded HTTP, time-based HTTP, and time based test increments.

No new string is required to identify the multi-threaded and time-based HTTP capabilities in DownloadTransports and UploadTransports because these features are just a generalization of the HTTP test which is already indicated by the sting “HTTP”.

Multi-Threaded HTTP test parameters

DownloadDiagnostics.ConcurrentSessions (unsigned int) and UploadDiagnostics.ConcurrentSessions (unsigned int) specify the number of concurrent TCP sessions (threads) to use for executing the multi-threaded HTTP test. When ConcurrentSessions is either 1 or 0, then a single session is used for HTTP tests. Note that the number of threads to use for an HTTP test can also be specified by the Threads (TH) bits in the TestResponse packet.

DownloadDiagnostics.TestBytesReceivedUnderFullLoading (unsigned int) and UploadDiagnostics.TestBytesSentUnderFullLoading (unsigned int) are the test traffic received and sent, respectively, in bytes (across all sessions) between the last BOMTime and the first EOMTime. Note DownloadDiagnostics.TestBytesReceived is the test traffic received in bytes (across all sessions) between the first BOMTime and the last EOMTime. Also, ConcurrentSessions*TestFileLength is the test traffic sent between the first BOMTime and the last EOMTime. That is, each session sends TestFileLength bytes.

DownloadDiagnostics.TotalBytesReceivedUnderFullLoading (unsigned int) and UploadDiagnostics.TotalBytesSentUnderFullLoading (unsigned int) is the total traffic received and sent, respectively, on the Ethernet interface in bytes between the last BOMTime and the first EOMTime. Note DownloadDiagnostics.TotalBytesReceived is the total traffic received on the Ethernet interface between the first BOMTime and the last EOMTime and UploadDiagnostics.TotalBytesSent is the total traffic sent across the Ethernet interface between the first BOMTime and last EOMTime.

DownloadDiagnostics.Result.{i}and UploadDiagnostics.Result{i} contains results for “each session” of a multi-threaded test including ROMTime, BOMTime, EOMTime, TestBytesReceived or TestBytesSent, TCPOpenRequestTime, and TCPOpenResponseTime. Note that the same variable names are also present at the DownloadDiagnostics and UploadDiagnostics level. At the DownloadDiagnostics and UploadDiagnostics level DownloadDiagnostics.ROMTime or UploadDiagnostics.ROMTime is the ROMTime of the last opened session, DownloadDiagnostics.BOMTime or UploadDiagnostics.BOMTime is the BOMTime of the last opened sessions, and DownloadDiagnostics.EOMTime or

UploadDiagnostics.EOMTime is the EOMTime of the first closed session of all the concurrent sessions.

Time-Based HTTP test parameters

Time-based HTTP tests perform a download or upload transaction for a pre-determined period of time (as opposed to terminating upon transfer of a file). As far as the receiver is concerned (i.e. the test server for an upload, or the BHR client for a download), it is interpreted no differently than a file transfer. However, the sender assures there are enough bytes in the transfer for it to take the pre-determined period of time (in seconds). A time-based HTTP transaction is performed when a particular DownloadURL (for a time-based download transaction) or UploadURL (for a time-based upload transaction) is specified as the DownloadDiagnostics.DownloadURL or UploadDiagnostics.UploadURL parameter, or whenever these special URLs are provided in a TestResponse packet. The special DownloadURL or UploadURL value is http://ServerHostname/timebasedmode_xx.txt where xx is a number indicating the number of seconds for the sender to perform the time-based download by the test server, or upload by the BHR client (e.g. http://ServerHostname/timebasedmode_30.txt for a 30 second test). Note that this particular URL is interpreted to be a time-based HTTP test at a TR-143 test server. If these values were to conflict with an existing file on a non-TR-143 HTTP server, then the actual file would be just downloaded or uploaded instead and the receiver would not know the difference. We propose that the download or upload be strictly ascii text so that an End of Transmission (EOT) character (Hex 0x04) can be used by the sender to indicate to the receiver that this is the last byte of the stream.

Incremental HTTP test parameters

The incremental test results mode is enabled (for a single threaded, multi-threaded, or time-based HTTP test) whenever DownloadDiagnostics.TimeBasedTestIncrements (unsigned int – seconds) or UploadDiagnostics.TimeBasedTestIncrements (unsigned int – seconds) is set to a value greater than zero.

If for example, DownloadDiagnostics.TimeBasedTestIncrements is set to 5, and a multi-threaded HTTP download test is performed, then starting from the last BOMTime of all the sessions, a timestamp is made every 5 seconds and DownloadDiagnostics.TestBytesReceivedUnderFullLoading is sampled at each timestamp. If the last timestamp occurs after the first EOMTime of all the sessions, then that last timestamp is set to that first EOMTime. Defining timestamp[0] as the last BOMTime of all the sessions, and timestamp[i] as the i-th timestamp from the last BOMTime (which should be BOMTime + i*5 seconds but may vary slightly from the ideal epochs), DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] is the i-th sample of DownloadDiagnostics.TestBytesReceivedUnderFullLoading taken at each time stamp. Note that if the first EOMTime from all the sessions were to occur at a non-multiple of 5 seconds (from the last BOMTime of all sessions), then the last

timestamp (call it timestamp[n]) would be set equal to the first EOMTime. Given these rules, we set DownloadDiagnostics.IncrementalResults{1}.IncBOMTime = timestamp[0] (i.e. the last BOMTime of all sessions), and DownloadDiagnostics.IncrementalResults{i}.IncEOMTime = timestamp[i], and DownloadDiagnostics.IncrementalResults{i}.IncBOMTime = DownloadDiagnostics.IncrementalResults{i-1}.IncEOMTime, for i= 1, 2, …, n. Also, we set DownloadDiagnostics.IncrementalResults{i}.IncTestBytesReceived = DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] (the i-th sampled value of TestBytesReceivedUnderFullLoading as it increments). Similarly, if UploadDiagnostics.TimeBasedTestIncrements is set to 5, the analogous rules apply for the upload case except DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] is replaced by UploadDiagnostics.TestBytesSentUnderFullLoading[i] and the corresponding upload variables in UploadDiagnostics.IncrementResults{i} substitute for those in DownloadDiagnostics.IncrementResults{i}.

[A.2] Below we present the client behaviors for the generalized HTTP download and upload tests.

Generalized HTTP Download Test:

1. Open (DownloadDiagnostics.ConcurrentSessions) TCP sockets for the multiple HTTP connections
    - Set the DownloadDiagnostics.Result.{i).TCPOpenRequestTime to the open request time for each TCP connection as it is opened
2. When TCP SYN-ACK is received for each TCP connection
    - Set DownloadDiagnostics.Result.{i).TCPOpenResponseTime to the current time (for each connection)
    - Set DownloadDiagnostics.TCPOpenRequestTime to DownloadDiagnostics.Result.{1).TCPOpenRequestTime, and DownloadDiagnostics.TCPOpenResponseTime to DownloadDiagnostics.Result.{1).TCPOpenResponseTime
3. After all TCP SYN-ACKs are received
    - Send a GET command on each connection to request the file (specified by DownloadDiagnostics.DownloadURL) and set the time of the i-th GET command sent to DownloadDiagnostics.Result.{i}.ROMTime
4. Upon receiving the HTTP (e.g. 200 OK) successful response for each GET:
    - Set DownloadDiagnostics.Result.{i).BOMTime to the current time value (for each connection)

- Start recording DownloadDiagnostics.Result.{i).TestBytesReceived for each connection as test traffic arrives. Also start recording DownloadDiagnostics.TestBytesReceived, the sum of all test bytes received on all the connections from the first BOM time until the last EOMTime, and DownloadDiagnostics.TotalBytesReceived, the total number of bytes received across the Ethernet interface between the first BOMTime and the last EOMTime.

5. Upon receiving the last HTTP (e.g. 200 OK) successful response
    - The current time is the last BOMTime value for all the connections. Set DownloadDiagnostics.BOMTime to the current DownloadDiagnostics.Result.{i).BOMTime, and set DownloadDiagnostics.ROMTime to DownloadDiagnostics.Result.{i).ROMTime for the same i.
    - Start recording the DownloadDiagnostics.TestBytesReceivedUnderFullLoading as the sum of test bytes received across all connections from the last BOMTime to the first EOMTime. Start recording DownloadDiagnostics.TotalBytesReceivedUnderFullLoading as the total number of bytes received over the Ethernet interface from the last BOMTime to the first EOMTime. *Note how TestBytesReceivedUnderFullLoading and TotalBytesReceivedUnderFullLoading are distinct from TestBytesReceived and TotalBytesReceived*.
6. If DownloadDiagnostics.TimeBasedTestIncrements > 0, then every DownloadDiagnostics.TimeBasedTestIncrements seconds (starting from the last BOMTime of all connections) do the following
    - Set DownloadDiagnostics.IncrementalResults{1}.BOMTime to the last BOMTime of all connections
    - At time DownloadDiagnostics.IncrementalResults{1}.BOMTime + DownloadDiagnostics.TimeBasedTestIncrements * j, timestamp the current time and set it equal to DownloadDiagnostics.IncrementalResults{j}.EOMTime for j = 1, 2, …, n
    - Set DownloadDiagnostics.IncrementalResults{j}.BOMTime = DownloadDiagnostics.IncrementalResults{j-1}.EOMTime for j = 1, 2, …, n
    - Set DownloadDiagnostics.IncrementalResults{j}.IncTestBytesReceived to the current value of DownloadDiagnostics.TestBytesReceivedUnderFullLoading (while it continues to increment)
    - Set DownloadDiagnostics.IncrementalResults{n}.EOMTime equal to the first EOMTime of all connections, where n is the last index for j (i.e. n is the final time slot index)
7. Upon receiving the last packet of data in each session

- If this is the last packet for the first session (connection) to complete its data transfer (i.e. the first EOMTime of all the connections), then set DownloadDiagnostics.EOMTime to the current time, and stop recording DownloadDiagnostics.TestBytesReceivedUnderFullLoading and DownloadDiagnostics.TotalBytesReceivedUnderFullLoading
- Set DownloadDiagnostics.Result.{i}.EOMTime for each connection and stop recording DownloadDiagnostics.Result.{i}.TestBytesReceived for each connection
- If this is the last packet for the last session (connection) to complete its data transfer (i.e. last EOMTime of all the connections), then stop recording DownloadDiagnostics.TestBytesReceived and DownloadDiagnostics.TotalBytesReceived

8. Wait for the server to close each connection or send a TCP RST if the timeout period (for connection close) is exceeded
9. Open a test results TCP connection to TestResultsTCPPort (Table 3). First construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse set to the values placed in DownloadDiagnostics.ROMTime, DownloadDiagnostics.BOMTime, DownloadDiagnostics.EOMTime, DownloadDiagnostics.TCPOpenRequest, DownloadDiagnostics.TCPOpenResponse as defined in steps (1), (2), (5) and (7). Set the TestBytesReceived field in the TestResult packet to the value placed in DownloadDiagnostics.TestBytesReceived if the test has a single thread (session) or in DownloadDiagnostics.TestBytesReceivedUnderFullLoading if the test is multi-threaded. Set the TotalBytesReceived field in the TestResult packet to the value placed in DownloadDiagnostics.TotalBytesReceived if the test has a single thread (session) or to DownloadDiagnostics.TotalBytesReceivedUnderFullLoading if the test is multi-threaded. Set the DownloadURL field to the URL that was downloaded. Send the TestResult packet to the test server. If Download.Diagnostics.TimeBasedTestIncrements > 0, construct a TestIncrement packet for each index, j, of DownloadDiagnostics.IncrementalResults{j}. For the j-th TestIncrement message, set IncBOMTime = DownloadDiagnostics.IncrementalResults{j}.BOMTime, IncEOMTime = DownloadDiagnostics.IncrementalResults{j}.EOMTime, IncTestBytesReceived = DownloadDiagnostics.IncrementalResults{j}.TestBytesRecevied. Send each TestIncrement packet to the test server. Also, send the upload test TestResult and TestIncrement packets to the test server if an upload test was also performed (i.e. DTE = 1 and UTE = 1). Construct the TestLocation packet and send it to the test server. Close the TestResult TCP connection.

Note that in step 6 above, when DownloadDiagnostics.TimeBasedTestIncrementsOffset > 0, we replace BOMTime (or the last BOMTime in a multi-threaded set of connections) with BOMTime + DownloadDiagnostics.TimeBasedTestIncrementsOffset. So

DownloadDiagnostics.TimeBasedTestIncrements does not actually start counting until BOMTime + DownloadDiagnostics.TimeBasedTestIncrementsOffset seconds have expired. This feature is used to remove the contribution from slow start in a throughput measurement if desired. Of course this will reduce the measured value of DownloadDiagnostics.TotalBytesReceivedUnderFullLoading by the number of bytes received during DownloadDiagnostics.TimeBasedTestIncrementsOffset.

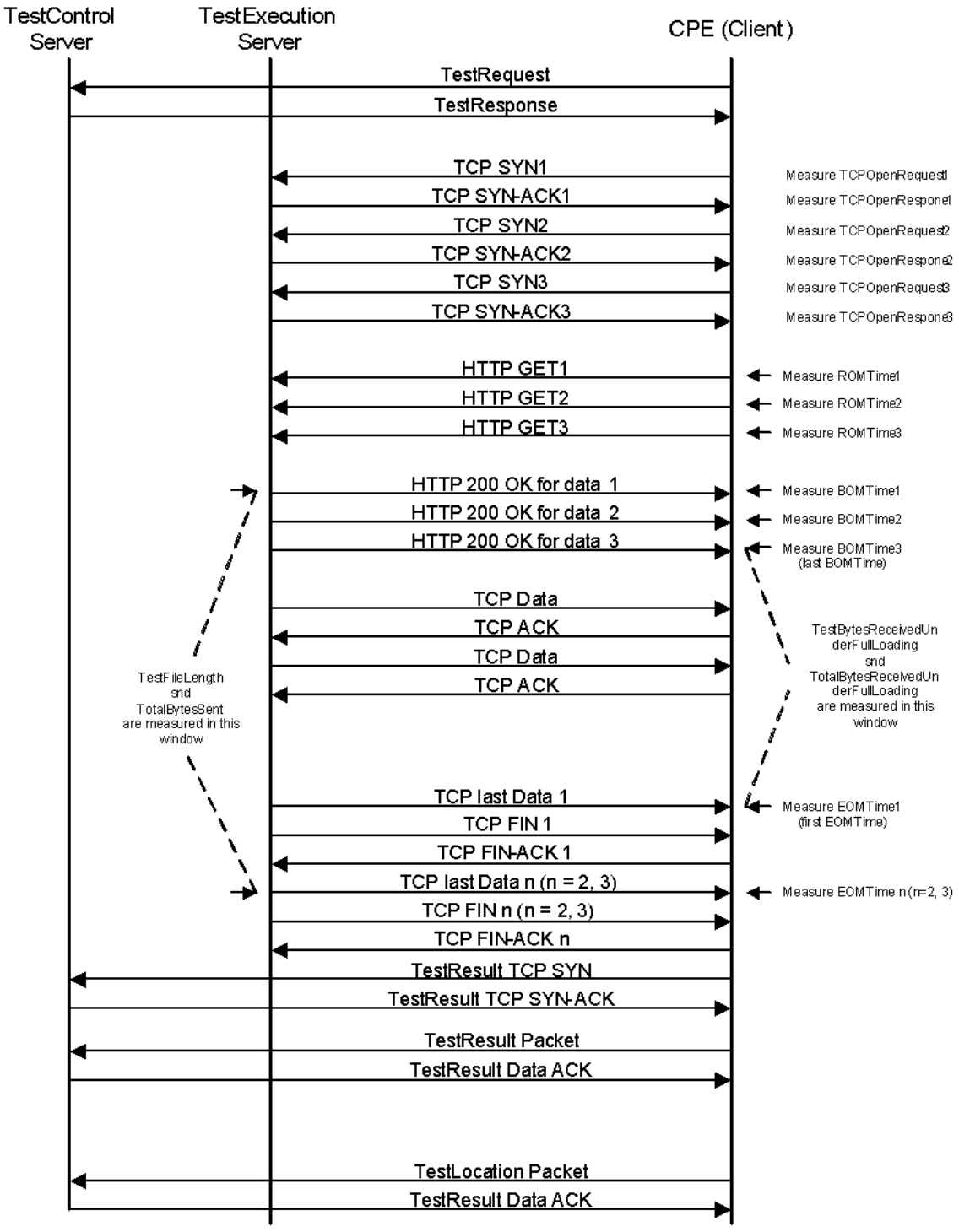

Figure 9 (Multi-Threaded HTTP Download)

Generalized HTTP Upload Test:

1. Open (UploadDiagnostics.ConcurrentSessions) TCP sockets for the multiple HTTP connections
    - Set the UploadDiagnostics.Result.{i).TCPOpenRequestTime to the open request time for each TCP connection as it is opened
2. When TCP SYN-ACK is received for each TCP connection
    - Set UploadDiagnostics.Result.{i).TCPOpenResponseTime to the current time (for each connection)
    - Set UploadDiagnostics.TCPOpenRequestTime to UploadDiagnostics.Result.{1).TCPOpenRequestTime, and UploadDiagnostics.TCPOpenResponseTime to UploadDiagnostics.Result.{1).TCPOpenResponseTime
3. After all TCP SYN-ACKs are received
    - Send a PUT command on each connection to request the sending of files (specified by UploadDiagnostics.UploadURL) and set the time of the i-th PUT command sent to UploadDiagnostics.Result.{i}.ROMTime
4. Upon receiving the TCP ACK for each PUT:
    - Set UploadDiagnostics.Result.{i).BOMTime to the current time value (for each connection)
    - Start recording UploadDiagnostics.Result.{i).TestBytesSent for each connection as test traffic is sent and UploadDiagnostics.TotalBytesSent, the total number of bytes sent across the Ethernet interface between the first BOMTime and the last EOMTime. Note that the transfer will send TestFileLength bytes on each connection to UploadURL unless UploadURL is set to http://ServerHostname/timebasedmode_xx.txt (where xx indicates the number of seconds for the transfer to occur) to denote time-based mode.
5. Upon sending the last PUT
    - The current time is the last BOMTime value for all the connections. Set UploadDiagnostics.BOMTime to the current UploadDiagnostics.Result.{i).BOMTime, and set UploadDiagnostics.ROMTime = UploadDiagnostics.Result.{i).ROMTime for the same i.
    - Start recording the UploadDiagnostics.TestBytesSentUnderFullLoading as the sum of test bytes sent across all connections from the last BOMTime to the first EOMTime. Start recording UploadDiagnostics.TotalBytesSentUnderFullLoading as the total number of bytes sent over the Ethernet interface from the last BOMTime to the first EOMTime. *Note how TestBytesSentUnderFullLoading and*

TotalBytesSentUnderFullLoading are distinct from FileTestLength and TotalBytesSent

6. If UploadDiagnostics.TimeBasedTestIncrements > 0, then every UploadDiagnostics.TimeBasedTestIncrements seconds (starting from the last BOMTime of all connections) do the following
   - Set UploadDiagnostics.IncrementalResults{1}.BOMTime to the last BOMTime of all connections
   - At time UploadDiagnostics.IncrementalResults{1}.BOMTime + UploadDiagnostics.TimeBasedTestIncrements * j, timestamp the current time and set it equal to UploadDiagnostics.IncrementalResults{j}.EOMTime, for j = 1, 2, …, n
   - Set UploadDiagnostics.IncrementalResults{j}.BOMTime = UploadDiagnostics.IncrementalResults{j-1}.EOMTime for j = 1, 2, …, n
   - Set UploadDiagnostics.IncrementalResults{j}.IncTestBytesSent to the current value of UploadDiagnostics.TestBytesSentUnderFullLoading (while it continues to increment)
7. Upon sending the last packet of data in each session, wait for each HTTP successful response (e.g. OK 200) from server
   - If this is the first EOMTime for all connections then set UploadDiagnostics.EOMTime to the current time, and stop recording UploadDiagnostics.TestBytesSentUnderFullLoading and UploadDiagnostics.TotalBytesSentUnderFullLoading
   - Set UploadDiagnostics.Result.{i}.EOMTime for each connection and stop recording UploadDiagnostics.Result.{i}.TestBytesSent for each connection
   - If this is the last packet for the last session (connection) to complete its data transfer (i.e. last EOMTime of all the connections), then stop recording UploadDiagnostics.TotalBytesSent
8. Initiate the closing of each of the HTTP connections
9. Open a test results TCP connection to TestResultsTCPPort (Table 3). First construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse set to the values that would be placed in UploadDiagnostics.ROMTime, UploadDiagnostics.BOMTime, UploadDiagnostics.EOMTime, UploadDiagnostics.TCPOpenRequest, UploadDiagnostics.TCPOpenResponse as defined in steps (1), (2), (5) and (7). Set the TestFileLength field in the TestResult packet to the value that would be placed in UploadDiagnostics.TestFileLength if the test has a single thread (session) or to UploadDiagnostics.TestBytesSentUnderFullLoading if the test is multi-threaded. Set the TotalBytesSent field in the TestResult packet to the value that would be placed in UploadDiagnostics. TotalBytesSent if the test has a single thread (session) or to UploadDiagnostics.TotalBytesSentUnderFullLoading if the test is

multi-threaded. Set the UploadURL field to the URL that was uploaded. Send the TestResult packet to the test server. If UploadDiagnostics.TimeBasedIncrements > 0, construct a TestIncrement packet for each index, j, of UploadDiagnostics.IncrementalResults{j}. For the j-th TestIncrement message, set IncBOMTime = UploadDiagnostics.IncrementalResults{j}.BOMTime, IncEOMTime = UploadDiagnostics.IncrementalResults{j}.EOMTime, IncTestBytesSent = UploadDiagnostics.IncrementalResults{j}.TestBytesSent. Send each TestIncrement packet to the test server. Construct the TestLocation packet and send it to the test server. Close the TestResult TCP connection.

Note that in step 6 above, when UploadDiagnostics.TimeBasedTestIncrementsOffset > 0, we replace BOMTime (or the last BOMTime in a multi-threaded set of connections) with BOMTime + UploadDiagnostics.TimeBasedTestIncrementsOffset. So UploadDiagnostics.TimeBasedTestIncrements does not actually start counting until BOMTime + UploadDiagnostics.TimeBasedTestIncrementsOffset seconds. This feature is used to remove the contribution from slow start in a throughput measurement if desired. Of course this will reduce the measured value of UploadDiagnostics.TotalBytesSentUnderFullLoading by the number of bytes received during UploadDiagnostics.TimeBasedTestIncrementsOffset.

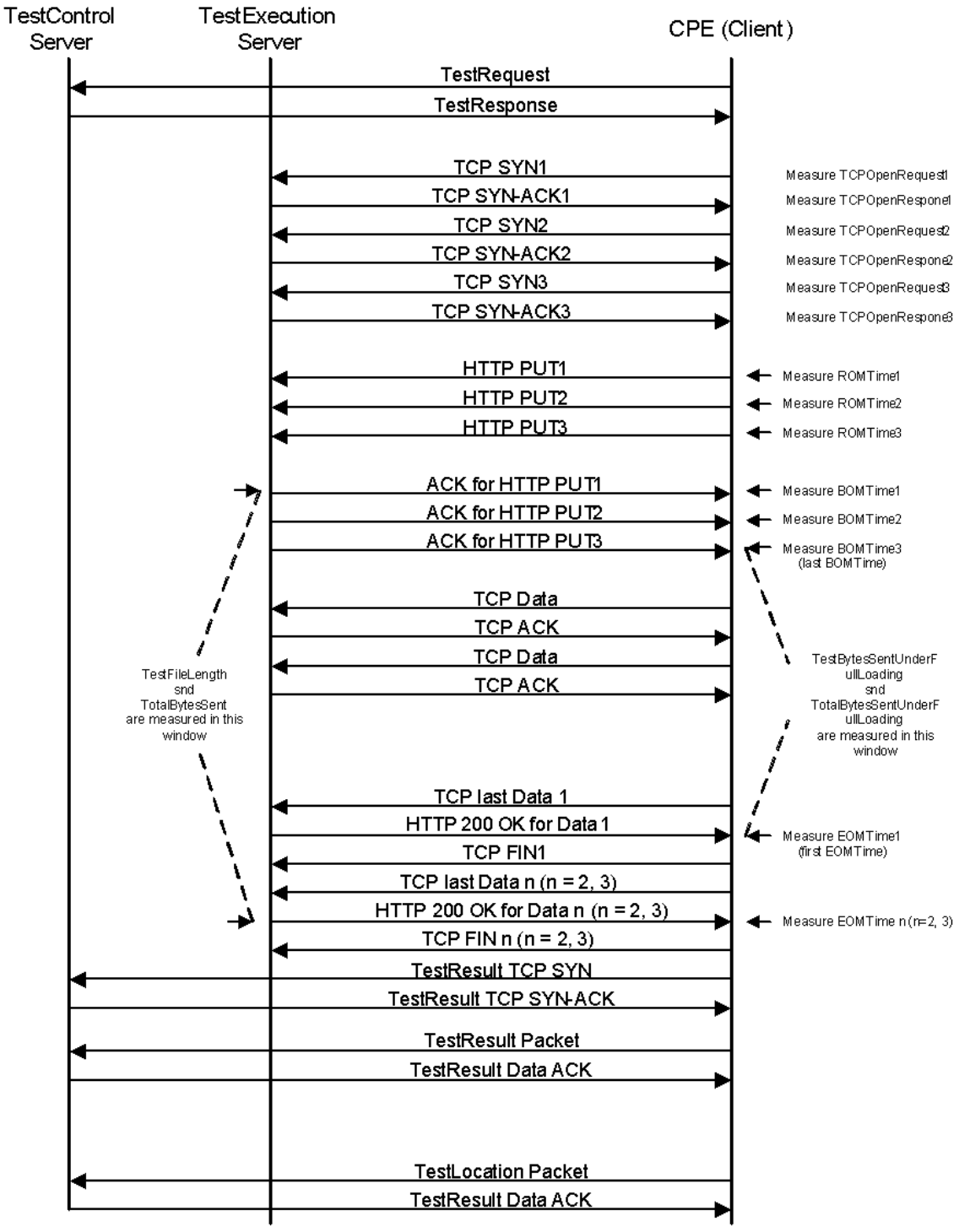

Figure 10 (Multi-Threaded HTTP Upload)

The TestIncrement packet has the following format. It is distinguished at the server by the DiagnosticState field being populated with the string “TestIncrement j”, where j is a number representing the index of the TestIncrement.

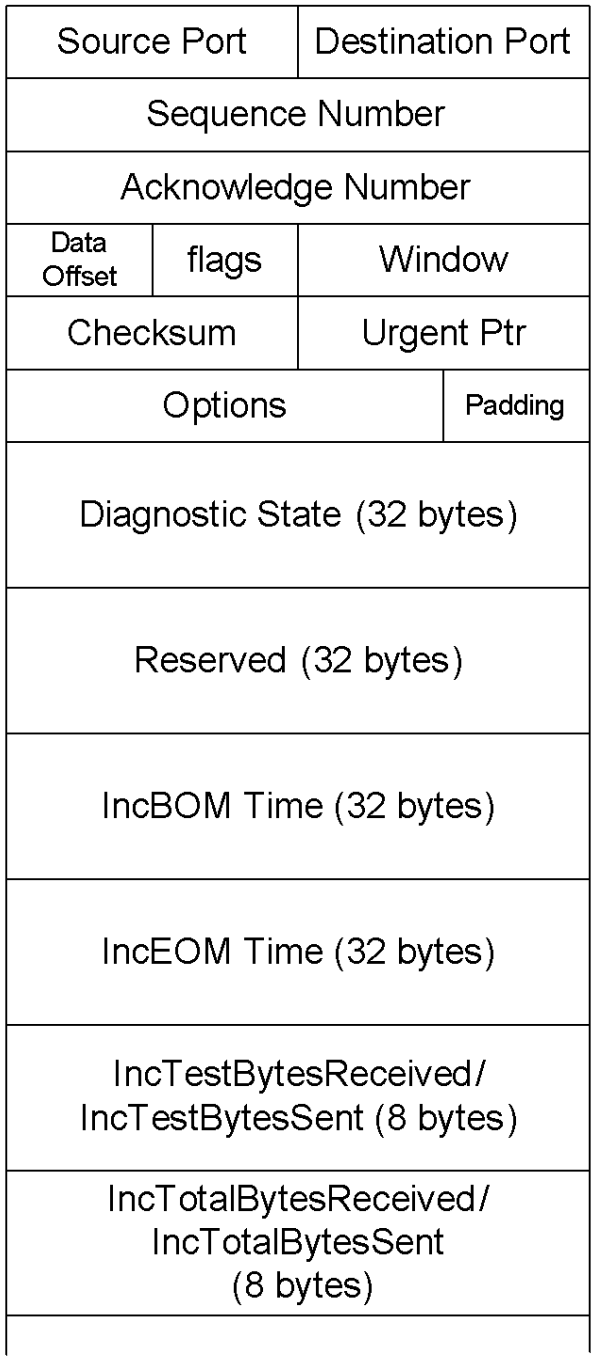

Figure 11 (TestIncrement packet format - Part 1)

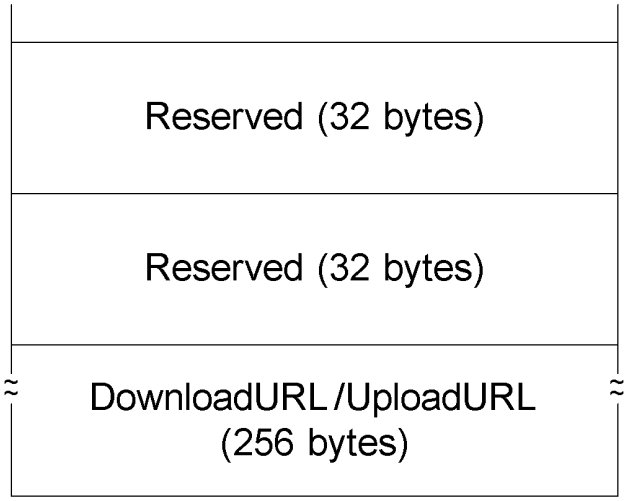

Figure 12 (TestIncrement - Part 2)

Appendix B – NDT Test Protocol

This section provides a detailed specification for the client behavior of the NDT test protocol which serves as an additional option to supplement the FTP and HTTP protocols defined in TR-143. It is anticipated that NDT will be included in the next revision of the TR-143 standard. Though the NDT protocol actually supports additional tests beyond upload and download throughput tests (e.g. a media interface test, simple firewall test, etc …), only the upload and download throughput tests are supported by the client behaviors described here. It is optional for the client to support the other NDT test types but mandatory for the client to support the NDT upload and download throughput tests. Three versions of the NDT client ("v3.1.4", "v3.4.1", and "v3.7.0") are defined with the version to be used selected by the NDTClientVersion configuration variable (thus NDTClientVersion can be set to any of those three string values to indicate which version of the NDT protocol the client will run). The version "v3.70" is defined in the specification provided at the link: https://code.google.com/p/ndt/wiki/NDTProtocol

[B.1] The device must support the following additional test configuration variables (in the path - InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.): NDTClientVersion (string – 16 bytes). Note that NDTClientVersion is a NULL terminated string that can be set to one of three values, "v3.1.4", "v3.4.1", or "v3.70". If

NDTClientVersion is set to an invalid value or is unassigned then the NDT test is not run. The NDT standard only supports a 10 sec throughput test option but this specification extends the capability of the standard to enable user selectable test duration. Moreover this specification supports multiple threads, time based test increments, and time-based test increments offset (as are specified for the HTTP throughput test in Appendix A – Extended Capabilities for the HTTP Test). The test duration parameter is conveyed to the test server in the DownstreamURL field of theTestRequest packets (though the parameters actually used for the test by the client are those provided from the server via the PT bit settings, TH bit settings, and DownstreamURL/UpstreamURL in the corresponding TestResponse packets). Since NDT now supports timebased test increments, the values forTimeBasedTestIncrements and TimeBasedTestIncrementsOffset parameters defined in Table 3 (TR-143 Test Control Variables – Phase 2), also apply to the NDT upload and download throughput tests. These parameters only take effect when they are assigned values greater than zero. Also note that when the version “v3.7.0” is used, that NDT messages are conveysed using JSON format. For all NDT versions, Web10G (or Web100 if Web 10G is not available) variables must be measured at the test server when the NDT throughput test is perfomed.

An NDT test can be requested by the CPE in the TestRequest packets, and is commanded by the test server in TestResponse packets when the PT bits are set to 2 (PT2 = 0, PT1 = 1, PT0 = 0). The NDT server hostname (or IP address) and the TCP port for NDT control are specified in the DownloadURL field of Test Request/Response packets. For this specification, an NDT throughput test always runs in both directions even though no UploadURL is specified, and so both the download and upload test results messages are returnedto the test server upon completion of the test. For an NDT test, the DownloadURL (i.e. NDT URL) variable has the format: “ndt://NDTServerHostname:NDTControlTcpPort”. NDT works by the client first opening a connection to a control tcp port (NDTControlTcpPort) which is used to accept commands from the NDT server (including which tcp ports to use for the upload and download tests).

Note NDT is a time based test. That is, the throughput test in each direction runs for 10 seconds by default. However, the NDT can run for different test durations. A more generalized example of a valid NDT URL as previously discussed in Section 3.2. "Throughput Test Initiation Mechanism (Phase 2)” is: “ndt://NDTServerHostname:NDTServerTCPPort: NDTTestDuration, where TestDuration (an unsigned integer) can specify an NDT test duration different than 10 seconds, Each of the paramters in the NDT URL, beyond NDTServerHostname, is optional as they each have default values. However, when the additional parameters are specified in the URL, they must be provided in the order indicated above. The default value for NDTServerTCPPort is tcp port 300 and the default value for NDTTestDuration is 10 (representing 10 seconds since the units for NDTTestDuration is seconds). So the URL “ndt://132.197.244.14” would indicate to the client to perform a 10 second NDT test (using the number of threads specified by the TH bits), using NDTServerTCPPort = 3001. The DownstreamURL “ndt://132.197.244.14:31” in the TestResponse packet

would indicate the client will perfom a 31 second NDT throughput test (using the number of threads specified by the TH bits). If a TimeBasedTestIncrements were set to 5 and TimeBasedTestIncrementsOffset were set to 1 during this test, then the CPE would measure throughput (snapshots) at the intervals from t = 1 sec to t = 6 sec, t = 6 sec to t = 11sec, t = 11 sec to t = 16 sec, t = 16 sec to t = 21 sec, t = 21 sec to t = 26 sec, and t = 26 sec to t = 31 sec. Note if all of the TimeBasedTestIncrements and TimeBasedTestIncrementsOffset parameters were identical with the DownstreamURL have a value of ndt://132.197.244.14:30" in the TestResponse packet (i.e. NDTTestDuration being 30 seconds), then the throughput snapshots would be measured at the same intervals except the last throughput snapshot would be measured from t = 26 sec to t = 30 sec so the last throughput snapshot interval can violate the TimeBasedTestIncrements value if TestIncrements does not divide evenly into TestDuration. Note these additional parameters defined in the NDT DownstreamURL (and the multiple number of threads specified by the TH bits) must be supported for all client versions specified here ("v3.1.4", "v3.4.1", and "v3.7.0").

[B.2] When NDTClientVersion is set to "v3.1.4", the client has the following behavior:

1. Open the control tcp connection to the NDT server on port NDTControlTcpPort (or port 3001 if NDTControlTcpPort is unassigned)

2. Wait for a (1 byte) Queue_Msg to be received from the NDT server (on NDTControlTcpPort) containing an ascii char representing the number of tests currently in the queue at the server. If any value other than ascii 0 is received, then wait for the next Queue_Msg. Note: Hex 30 (ascii 0) indicates zero tests in queue so when Hex 30 is received in the Queue_Msg, run the throughput test now.

3. Wait for TestPort_Msg packet (on NDTControlTcpPort) containing TCP ports to use for upstream/downstream tests. The TestPort_Msg contains an ascii string containing two substrings delimited by a whitespace character (Hex 20). The first substring denotes the upstream tcp port and the second substring is the downstream tcp port to be used for tests in the respective directions.

4. Open a tcp connection to the NDT server on the downstream test port obtained from step (3).

5. Accept data from the downstream test port tcp connection until the server closes the connection. This tcp connection is not the actual downstream test. It represents a preliminary connection where the NDT server provides ascii representations of the server IP address and client IP address it detected, both in dot-decimal form. Discard this data. Upon receiving a FIN from the server on the downstream test port, close the downstream test port tcp connection (note: the control tcp connection is still open).

6. Open a tcp connection to the NDT server on the upstream test port obtained from step (3). Use the SYN and SYN-ACK time for the connection open request and response and set to UploadDiagnostics.TCPOpenRequest and UploadDiagnostics.TCPOpenResponse. Note that if a multithreaded test is being performed (i.e. TH bits set to a value greater than 1), then open the number of connections coded in the TH bits starting on the upstream test port obtained from step 3 and increment the tcp port by 1 for each additional connection. For the multithreaded case, measure UploadDiagnostics.TCPOpenRequest and UploadDiagnostics.TCPOpenResponse using SYN and SYN-ACK of the first tcp connection opened.

7. Wait for TestPort_Msg packet (on NDTControlTcpPort) before sending data upstream. This TestPort_Msg is used as a start indicator for the upstream test. Upon receiving the TestPort_Msg timestamp the current time and set it equal to the upload diagnostics ROM_TIME value. Then send data upstream for 10 seconds (or NDTTestDuration seconds if a value different then 10 was received in the DownstreamURL of the TestResponse message), time stamping the first and last packets sent upstream on each test tcp connection opened and setting the upload diagnostics BOM_Time to the timestamp of the first packet sent during the test (after all the connections in a multithreaded test have been opened) and EOM_Time to the timestamp of the last packet sent prior to NDTTestDuration being reached. Note EOM_Time should be approximately BOM_Time + NDTTestDuration seconds (the difference being the time it takes to forward all the queued data from the send buffer). Note that for the either a single threaded or multithreaded NDT test EOMTime – BOMTime = PeriodOfFullLoading (with BOMTime and EOMTime defined as above) so set PeriodOfFullLoading to this difference. Measure the total number of bytes sent upstream and set that value to TestFileLengh. If TimeBasedTestIncrements and/or TimeBasedTestIncrementsOffset are greater than zero, then perform the throughput snapshots during intervals of TimeBasedTestIncrements starting from TimeBasedTestIncrementsOffset from BOMTime. Also, increment the TotalBytesUpstream variable by the number of total bytes measured on the WAN uplink during the test. Close the connection(s) after data has been sent upstream for NDTTestDuration seconds.

8. Open a tcp connection to the NDT server on the downstream test port obtained from step (3). Use the SYN and SYN-ACK time for the connection open request and response and set to DownloadDiagnostics.TCPOpenRequest and DoanloadDiagnostics.TCPOpenResponse. Note that if a multithreaded test is being performed (i.e. TH bits set to a value greater than 1), then open the number of connections coded in the TH bits starting on the downstream test port obtained from step 3 and increment the tcp port by 1 for each additional connection. Also for the multithreaded case, measure UploadDiagnostics.TCPOpenRequest and UploadDiagnostics.TCPOpenResponse using SYN and SYN-ACK of the first tcp connection opened.

9. Wait for TestPort_Msg packet (on NDTControlTcpPort) before receiving data in the downstream direction. This TestPort_Msg is used as a start indicator for the downstream test. Upon receiving the TestPort_Msg timestamp the current time and set it equal to the download diagnostics ROM_TIME value. Now receive data from the NDT server (test will run for NDTTestDuration seconds as specified in the DownstreamURL or 10 seconds by default if NDTTestDuration is not specified) until the NDT server closes the connection, time stamping the first packet and last packet received prior to NDTTestDuration expiring and setting the download diagnostics values for BOM_Time (timestamp of first packet received) and EOM_Time (timestamp of last packet received prior to NDTTestDuration seconds) to those timestamps, respectively. For a multithreaded test, all connections should be kept open until after NDTTestDuration has been reached. Measure the total number of bytes received and set that value to TestBytesReceived. If a multithreaded NDT download test is performed then set PeriodOfFullLoading to the duration in which traffic is received and all of the connections are simultaneously opened. If TimeBasedTestIncrements and/or TimeBasedTestIncrementsOffset are greater than zero, then perform the throughput snapshots during intervals of TimeBasedTestIncrements starting at TimeBasedTestIncrementsOffset from BOMTime. Also, increment the TotalBytesDownstream variable by the number of total bytes measured on the WAN uplink during the test. The NDT server will close the connection(s) after test data has been sent downstream for NDTTestDuration seconds.

10. Continue to receive data on the NDTControlTcpPort until the server closes that connection (marking the end of the test). This additional data on NDTControlTcpPort is the test results being provided to the client from the NDT server. This test result data is just discarded since the BHR has already performed its own measurements for the NDT test.
11. Open a test results TCP connection to the ServerIP address on port TestResultsTCPPort (Table 3). First construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse populated based on the results measured for the NDT downstream test. Send the TestResult packet. If test increments were performed then sent the required number of TestIncrement messages as defined in Figure 11 (TestIncrement packet format - Part 1) for the downstream test. Next construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse populated based on the results measured for the NDT upstream test. Send the TestResult packet Send the TestResult packet. If test increments were performed then sent the required number of TestIncrement messages as defined in Figure 11 (TestIncrement packet format - Part 1) for the upstream test.

[B.3] When NDTClientVersion is set to “v3.4.1”, a stream messaging protocol is used between the NDT server and client across the NDTControlTcpPort. Once the control tcp connection is opened, the client and server exchange commands on the stream per the format below. Each message has the format: Msg_Command (1-byte), Msg_Length (2-bytes), and Msg_Data (Msg_Length-bytes).

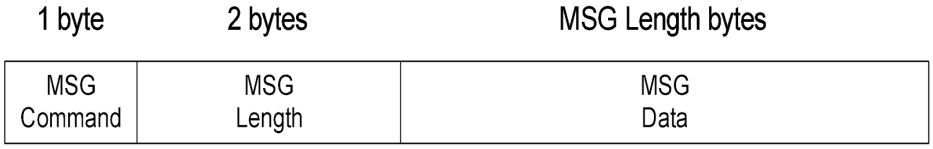

Figure 13 (NDT message format)

The following message commands are defined:

COMM_FAILURE = 0x00;
SRV_QUEUE = 0x01;
MSG_LOGIN = 0x02;
TEST_PREPARE = 0x03;
TEST_START = 0x04;
TEST_MSG = 0x05;
TEST_FINALIZE = 0x06;
MSG_ERROR = 0x07;
MSG_RESULTS = 0x08;

[B.4] When NDTClientVersion is set to “v3.4.1”, the client has the following behavior:

1. Open the control tcp connection to the NDT server on port NDTControlTcpPort (or port 3001 if NDTControlTcpPort is unassigned)

2. Send the MSG_LOGIN (0x02) message having length 1 (0x0001) and a msg data value of 0x06 to request upload and download tests (i.e. send 0200106 to the server)

3. Receive 13 byte message dummy message from the server (used to remove older clients)

4. Wait for SRV_QUEUE (0x01) message having length 1 (0x0001) indicating the number of tests in the NDT server queue. For example, receive a message of the form 010001yy, where yy is an ascii representation of the number of jobs in the NDT server queue. The number of jobs in the queue is called the “wait flag” (e.g. 01000132 indicates "wait flag" = 2, as 32 represents 2 in ascii)

5. Keep accepting SRV_QUEUE (0x01) messages until the "wait flag" is zero (e.g. 01000130 indicates “wait flag” = 0, as 30 represents 0 in ascii). After

SRV_QUEUE (0x01) with "wait flag" equal zero is received, wait for MSG_LOGIN (0x02) message. This MSG_LOGIN (0x02) message provides the NDT version information string (e.g. 02000776332e342e34a is a MSG_LOGIN msg, having length 7, indicating "v3.4.4a"). *Note the v3.4.1 works with all later releases of the server to date so one may see an NDT server version later than v3.4.1.*

6. Wait for MSG_LOGIN (02) containing a string that indicates which tests were selected, with each test separated by a space charater value of '20' (e.g. 020006203820322034 suggests a TEST_SFW (0x08), TEST_C2S (0x02), and TEST_S2C (0x04), where the 38 is ascii 8, 32 is ascii 2, and 34 is ascii 4). TEST_C2S stands for client to server (or upstream) test, TEST_S2C stands for server to client (or downstream) test, and TEST_SFW stands for a Simple Firewall Test. The other test type option is TEST_MID (00x01). Since we sent 0x06 to the server in step (2) to select the only the upstream and downstream tests, we expect to receive 0200420322034 which suggests only TEST_C2S (0x02) and TEST_S2C (0x04) tests were selected)

7. Wait for TEST_PREPARE (0x03) message having length 4 which provides the upstream test port coded in ascii (e.g. 0300433303032 means perform upstream TEST_C2S test to port 3002)

8. Open tcp connection to upstream test port obtained from Step (7). Use the SYN and SYN-ACK time for the connection open request and response and set to DownloadDiagnostics.TCPOpenRequest and DoanloadDiagnostics.TCPOpenResponse. Note that if a multithreaded test is being performed (i.e. TH bits set to a value greater than 1), then open the number of connections coded in the TH bits starting on the upstream test port obtained from step 7 and increment the tcp port by 1 for each additional connection. Also for the multithreaded case, measure UploadDiagnostics.TCPOpenRequest and UploadDiagnostics.TCPOpenResponse using SYN and SYN-ACK of the first tcp connection opened.

9. Wait for TEST_START (0x04) message (on NDTControlTcpPort) having message length zero. Timestamp the TEST_START message arrival and set the ROM_TIME in the upload diagnostics to this timestamp value. Then send data upstream for 10 seconds (or NDTTestDuration seconds if a value different then 10 was received in the DownstreamURL of the TestResponse message), time stamping the first and last packets sent upstream on each test tcp connection opened and setting the upload diagnostics BOM_Time to the timestamp of the first packet sent during the test (after all the connections in a multithreaded test have been opened) and EOM_Time to the timestamp of the last packet sent prior to NDTTestDuration being reached. For a multithreaded test, all connections should be kept open until after NDTTestDuration hs been reached. Note

EOM_Time should be approximately BOM_Time + NDTTestDuration seconds (the difference being the time it takes to forward all the queued data from the send buffer). Note that for either a single threaded or multithreaded NDT test EOMTime – BOMTime = PeriodOfFullLoading (since it is expected all connections will be maintained until after NDTTestDuration has been reached) so set PeriodOfFullLoading to this difference. Measure the total number of bytes sent upstream and set that value to TestFileLengh. If TimeBasedTestIncrements and/or TimeBasedTestIncrementsOffset are greater than zero, then perform the throughput snapshots during intervals of TimeBasedTestIncrements starting at TimeBasedTestIncrementsOffset from BOMTime. Also, increment the TotalBytesUpstream variable by the number of total bytes measured on the WAN uplink during the test. Close the connection(s) after data has been sent upstream for NDTTestDuration seconds.

10. Wait for TEST_MSG (0x05) message (on NDTControlTcpPort) which contains a string formatted upstream speed test result in Kb/s as the message data

11. Wait for TEST_FINALIZE (0x06) message (on NDTControlTcpPort) having message date length of zero (e.g. 060000).This message terminates the upstream test portion of the NDT test.

12. Wait for TEST_PREPARE (0x03) message (on NDTControlTcpPort) having length 4 which provides the downstream test port coded ascii (e.g. 0300433303033 means perform downstream (TEST_S2C) test to port 3003)

13. Open tcp connection to downstream test port obtained from Step (12).). Use the SYN and SYN-ACK time for the connection open request and response and set to DownloadDiagnostics.TCPOpenRequest and DoanloadDiagnostics.TCPOpenResponse. Note that if a multithreaded test is being performed (i.e. TH bits set to a value greater than 1), then open the number of connections coded in the TH bits starting on the downstream test port obtained from step 12 and increment the tcp port by 1 for each additional connection. Also for the multithreaded case, measure UploadDiagnostics.TCPOpenRequest and UploadDiagnostics.TCPOpenResponse using SYN and SYN-ACK of the first tcp connection opened.

14. Wait for TEST_START (0x04) message (on NDTControlTcpPort) having message length zero. Timestamp the TEST_START message arrival and set the ROM_TIME in the download diagnostics to this timestamp value. Receive downstream data from the NDT server (on the downstream test port or test ports for the multithreaded case) time stamping the first packet received and the last packet received prior to NDTTestDuration being reached setting the download diagnostics for BOM_Time (timestamp of first packet received) and EOM_Time (timestamp of last packet received prior to NDTTestDuration seconds) to those

timestamps, respectively. For a multithreaded test, all connections should be kept open until after NDTTestDuration has been reached so EOMTime – BOMTime = PeriodOfFullLoading. Measure the total number of bytes received and set that value to TestBytesReceived. If a multithreaded NDT download test is performed then set PeriodOfFullLoading to the duration in which traffic is received and all of the connections are simultaneously opened. If TimeBasedTestIncrements and/or TimeBasedTestIncrementsOffset are greater than zero, then perform the throughput snapshots during intervals of TimeBasedTestIncrements starting at TimeBasedTestIncrementsOffset from BOMTime. Also, increment the TotalBytesDownstream variable by the number of total bytes measured on the WAN uplink during the test. The NDT server will close the connection(s) shortly after test data has been sent downstream for NDTTestDuration seconds. Complete the closing of the downstream test port tcp connectiona upon receiving a FINs from the server

15. Wait for TEST_FINALIZE (0x06) message (on NDTControlTcpPort) having length zero (e.g. 060000). Several TST_MSG (0x 05) will arrive before the TEST_FINALIZE message (each containing Web100 or Web10G data), but just continue to read and discard the TST_MSG messages until the TEST_FINALIZE message arrives

16. Wait for TEST_RESULTS (0x08) message and its associated message data. Discard the test results since we have already performed our measurements in step (14).

17. Close the NDTControlTcpPort control tcp connection upon recevieving a FIN from the NDT server.

18. Open a test results TCP connection to the ServerIP address on port TestResultsTCPPort (Table 3). First construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse populated based on the results measured for the NDT downstream test. Send the TestResult packet. If test increments were performed then sent the required number of TestIncrement messages as defined in Figure 11 (TestIncrement packet format - Part 1) for the downstream test. Next construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse populated based on the results measured for the NDT upstream test. Send the TestResult packet Send the TestResult packet. If test increments were performed then sent the required number of TestIncrement messages as defined in Figure 11 (TestIncrement packet format - Part 1) for the upstream test.

[B.5] When NDTClientVersion is set to “v3.7.0”, a stream messaging protocol is used between the NDT server and client across the NDTControlTcpPort. Once the control tcp connection is opened, the client and server exchange commands on the stream per the format illustrated in Figure 13 (NDT message format). One differenece between “v3.4.1” and “v3.7.0”, however, is that for “v3.7.0” the messages are transmitted using JSON format. Also, for “v3.7.0”, the number of supported commands is extended beyond those depicted in section [B.3] for “v3.4.1”.

The following additional message commands are defined for “v.3.7.0”:

MSG_LOGOUT = 0x09;
MSG_WAITING = 0x10;
MSG_EXTENDED_LOGIN = 0x11;

[B.6] When NDTClientVersion is set to “v3.7.0” the client has the following behavior (as specified in https://code.google.com/p/ndt/wiki/NDTProtocol with some extension to support multiple threads and TimeBasedTestIncrements):

1. Open the control tcp connection to the NDT server on port NDTControlTcpPort (or port 3001 if NDTControlTcpPort is unassigned)

2. Send the MSG_EXTENDED_LOGIN (0x11) message having length 1 plus the number of bytes required to send the version number string. The extra one byte in the length is required to include the TESTS field and the TESTS field shall have a data value of 0x16 (i.e. 22 decimal) to request upload and download tests plus indicate that the device supports TEST_STATUS messages. Note that the 0x16 is the logical bitwise OR of 1L << 1 = 2 (indicating a client to server or upload throughput test is to be performed), 1L << 2 = 4 (indicating a server to client or download throughput test is to be performed) , and 1L << 4 = 16 (indicating the TEST_STATUS message is supported by this device). So, for example, the “v3.7.0” client device could send the following MSG_EXTENDED_LOGIN message using JSON formatting (after opening a connection on the NDTControlPort): 1100061676332e3730 (where 11 indicates the MSG_EXTENDED_LOGIN message, 0006 indicates the length (this field is 2 bytes long), 16 indicates an upload test and download test are to be performed and TEST_STATUS message is supported, and 76332e3730 is the string “v3.70”.

3. Wait to receive 13 byte message dummy message from the server (used to remove older clients). The 13 byte dummy message is usually set to the character sequence ‘1’,‘2’,‘3’,‘4’,‘5’,‘6’, ‘ ‘, ‘6’, ‘5’, ‘4’, ‘3’, ‘2’, ‘1’.

4. Wait for SRV_QUEUE (0x01) message having length 1 (0x0001) and having a data value of character ‘0’ in the SRV_QUEUE data field (i.e. 01000130). Note

that other values the SRV_QUEUE message can have in the data field are “9977” (i.e. a SRV_QUEUE message of 01000439393737) indicating a server fault, or “9988” (i.e. a SRV_QUEUE message of 01000439393838) indicating the server is busy, or “9990” (i.e. a SRV_QUEUE message of 010000439393930) indicating the server is attempting to verify the client is alive, or a value of “N” (where N is a string representation of a number) indicating the estimated number of remaining minutes the client must wait for the test to begin. Note that whenever the server sends a SRV_QUEUE message having value “9990” (i.e. a SRV_QUEUE message of 010000439393930), the client must respond with a MSG_WAITING message (e.g. 10000130).

5. After receiving the SRV_QUEUE message having the value of character ‘0’ described in step (4), wait for a MSG_LOGIN message from the server which provides the version of the server. Check the server version in this MSG_LOGIN message to ensure that the server version is greater than or equal to version 3.3.12. If the server version is less than version 3.3.12, then drop the NDTControl connection and abandon the test. Note that a server must be atleast version 3.3.12 to support a “v3.7.0” client.

6. Wait for a MSG_LOGIN message from the server which includes the test ids (i.e. test types) that the server will run. Note that these test id’s should only include the upload throughput (or client to server test id = 2), and/or download throughput (or server to client test id = 4) test id values since these were the only two test type that were requested in step (2). Note that the test ids are encoded as a space separated string in this MSG_LOGIN message from the server (as opposed to the bit encoding used by the client to request each test types in step (2)). So the server would indicate it will perfrom an upload (i.e. C2S) test followed by a download (i.e. S2C) test by including the following string in the data field of the MSG_LOGIN message: ‘2’, ‘ ‘, ‘4’ (where the order of the characters indicates the order in which the tests will occur). So the complete MSG_LOGIN message for this scenario of an upload test followed by a download test would be 020003322034. If by some chance the server includes other test ids (i.e. to suggest other test types be performed), than drop the connection and abandon the test.

7. The client then runs a test for each test id supplied in the MSG_LOGIN message of step (6). The client here deviates from that described in https://code.google.com/p/ndt/wiki/NDTProtocol in that it runs for a test duration specified in the DownstreamURL (which may differ from the standard 10 sec test described in https://code.google.com/p/ndt/wiki/NDTProtocol), it may also run with multiple threads if the TH bits are encoded to a value greater than 1, and it may perform time based test increments (when TimeBasedTestIncrements of Table 3 is greater than 0) with a time based test increment offset (when TimeBasedTestIncrementsOffset of Table 3 is greater than 0). When the S2C (i.e. download NDT test) and C2S (i.e. upload NDT test) are run (each for

NDTTestDuration seconds) the values for ROMTime, BOMTime and EOMTime are computed in a manner identical to the "v3.4.1" NDT client.

8. Upon completion of the upstream and downstream NDT tests, open a test results TCP connection to the ServerIP address on port TestResultsTCPPort (Table 3). First construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse populated based on the results measured for the NDT downstream test. Send the TestResult packet. If test increments were performed then sent the required number of TestIncrement messages as defined in Figure 11 (TestIncrement packet format - Part 1) for the downstream test. Next construct a TestResult packet (Figure 4) with ROMTime, BOMTime, EOMTime, TCPOpenRequest, and TCPOpenResponse populated based on the results measured for the NDT upstream test. Send the TestResult packet Send the TestResult packet. If test increments were performed then sent the required number of TestIncrement messages as defined in Figure 11 (TestIncrement packet format - Part 1) for the upstream test. Finally send a TestLocation message.

Appendix C – DNS Response Time Test Protocol

This section provides a specification for the client behavior of the DNS response time test protocol which serves as an additional test type option. This specification is based on Namespace Lookup Diagnostics profile of TR-157.

[C.1] We propose adding NSLookUpDiagnostic at the path location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.NSLookUpDiagnostic for the purpose of incorporating it to the TR-143 test suite. The table is shown below:

.NSLookupDiagnostics.	object	-	This object defines access to an IP-layer NSLookup test for the specified IP interface. When initiated, the NS Lookup test will contact *DNSServer* and look up *HostName* *NumberOfRepetitions* times. There will be a *Result* instance for each time the device performs a DNS lookup, which is determined by the value of *NumberOfRepetitions*. Any previous *Result* instances are removed when a new test is initiated.	-
DiagnosticsState	string	W	Indicates availability of diagnostic data. Enumeration of: *None*	-

			Requested *Complete* *Error_DNSServerNotResolved* (Unable to resolve DNSServer Name) *Error_Internal* *Error_Other* If the ACS sets the value of this parameter to *Requested*, the CPE MUST initiate the corresponding diagnostic test. When writing, the only allowed value is *Requested*. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to *Requested*. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the test is completed, the value of this parameter MUST be either *Complete* (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than *Complete*, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to *None*. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to *None*. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to *None*. While the test is in progress, setting this parameter to *Requested* (and possibly modifying	

			other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameter	
Interface	string(256)	W	The value MUST be the full path name of a table row. This parameter specifies the IP-layer interface over which the test is to be performed (i.e. the source IP address to use when performing the test). If an empty string is specified, the CPE MUST use its routing policy (Forwarding table entries), if necessary, to determine the appropriate interface.	-
Hostname	string(256)	W	Specifies the Host Name that NS Lookup is to look for. The current domain name MUST be used unless the name is a fully qualified name.	-
DNSServer	string(256)	W	Specifies the DNS Server name or IP address that NS Lookup is to use for the lookup. The name of this server will be resolved using the default DNS server unless an IP address is provided. If an empty string is specified, the device's default DNS server will be used. If this parameter is set to the value “Internal” or the IP address of the internal DNS server which services the LAN ports on this device (e.g. 192.168.1.1), then the DNS server internal to this device that services the LAN ports of the device shall be used (with all of its internal caching, etc … as would be applied to service an external device connected to one of the device LAN ports).	-
Timeout	unsignedInt	W	Timeout in *milliseconds* that indicates that a request has failed.	-
NumberOfRepetitions	unsignedInt	W	The number of times the device SHOULD repeat the execution of the NSLookup using the same input parameters. If the diagnostics test fails the CPE MAY terminate the test without completing the full number of repetitions. Each repetition will use a Result instance to hold the NSLookup result data. The parameter can be set to a value between 1 and 8 inclusive.	-
SuccessCount	unsignedInt	W	Number of successfully executed repetitions.	-
ResultNumberOfEntries	unsignedInt	W	Total number of Result entries from the most recent invocation of the test.	-
.NSLookupDiagnostics.Result.{i}.	object	-	Results from the most recent invocation of the test, one instance per repetition.	-
Status	string	-	Result Parameter to represent whether the NSLookup was successful or not.	-

			Errors for individual Result instances do not get bubbled up to *.NSLookupDiagnostics.DiagnosticsState*. A failure on a specific attempt does not mean that the overall test failed, but a failure on all attempts means that *.NSLookupDiagnostics.DiagnosticsState* SHOULD be *Error_Other*. Enumeration of: *Success* *Error_DNSServerNotAvailable* *Error_HostNameNotResolved* *Error_Timeout* *Error_Other*	
AnswerType	string	-	Result parameter to represent whether the answer is Authoritative or not. Enumeration of: *None* (Indicates that the NS Lookup failed to find the host.) *Authoritative* *NonAuthoritative*	-
HostNameReturned	string(256)	-	Result parameter to represent the fully qualified name for the Host Name in the calling parameter (e.g. HostName.DomainName); if no response was provided, then this parameter is an empty string.	-
IPAddresses	string	-	Comma-separated list (maximum 10 items) of IPAddresses. Indicates the IP Address results returned by the NS Lookup; if no response was provided, then this parameter is an empty string.	-
DNSServerIP	string	-	Result parameter to represent the actual DNS Server IP address that the NS Lookup used.	-
ResponseTime	unsignedInt	-	Response time (for the first response packet) in *milliseconds*, or 0 if no response was received.	-

Table 5 - NSLookup Object Table (TR-157)

Because the DNS response time test is actually not a throughput test, per se, it requires some modifications to the field definitions of the TestRequest/TestResponse and TestResult messages (while maintaining the same structure). For a DNS response time test, the TestRequest/TestResponse messages will have the following format:

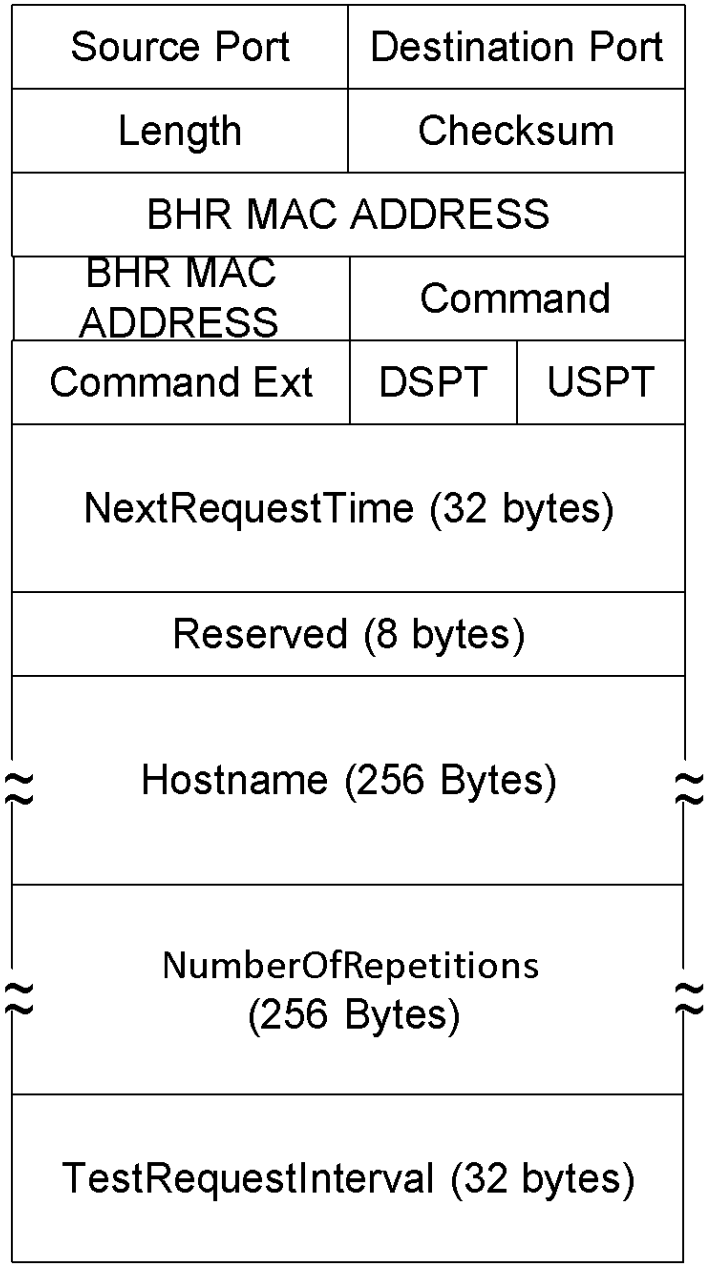

Figure 14 – TestRequest packet for DNS Response Test

The DownloadURL field is replaced by a Hostname field and the UploadURL field is replaced by the NumberOfRepetitions in both the TestRequest and TestResponse when the PT bits are coded to specify a DNS response time test (i.e. PT2 = 0, PT1 = 1, PT0 = 1). The Hostname field holds a string formatted representation of the Hostname that the BHR client should perform the lookup for when the test is performed. Since the lookup can be performed multiple times for a single test (see Table 5), another field called

NumberOfRepetitions is defined in place of the UploadURL field. The NumberOfRepetitions field contains a string representation of the number of times the loopkup should be performed for the test and can be set to a value between 1 and 8 inclusive. The Hostname and NumberOfRepetitions fields essentially replace the Hostname and NumberOfRepetitions parameters in NSLookupDiagnostics table (Table 5) with the remaining configuration parameters (e.g. DNSServer, Interface, Timeout, etc …) provisioned in the BHR by the ACS via TR-069. Note that the DNS Response time test is only performed when the DTE bit in the TestResponse packet is set to 1. Upon completion of the DNS response test, TestResult packets are used to communicate the DNS response time results back to the test server. Some of the fields in the TestResult packet are also re-defined for a DNS response test as shown in the figure below. Note that since a DNS response time test can have multiple repetitions (as specified by the NumberOfRepetitions field in the TestRequest/TestResponse packet), a separate TestResult packet is forwarded from the BHR to the test server for each repetition of a given test to provide a unique Status, IPAddress, DNSServerIP, and ResponseTime value for each repetition within a test. A TestLocation packets is sent following all the TestResult packets for each repetition of the DNS response time test. Note the DiagnosticState field value is the same for each repetition though the Status, IPAddress, DNSServerIP, and ResponseTime are unique per repetition. It is recommended that the NumberOfRepetitions be kept in the range from 1 to 8 though more repetitions are possible. To distinguish the DNS response time test from other test types (e.g. throughput tests) at the test server, each string placed in the DiagnosticState field for a DNS response time test will be pre-pended with the substring “DNS_” so a DiagnosticState value of “Completed” shall be coded as “DNS_Completed” by the BHR. A DiagnosticState value of “Error_DNSServerNotResolved” shall be coded as “DNS_Error_DNSServerNotResolved” and so on. This enables the test server to determine the TestResult packet is in fact communicating a DNS response test result.

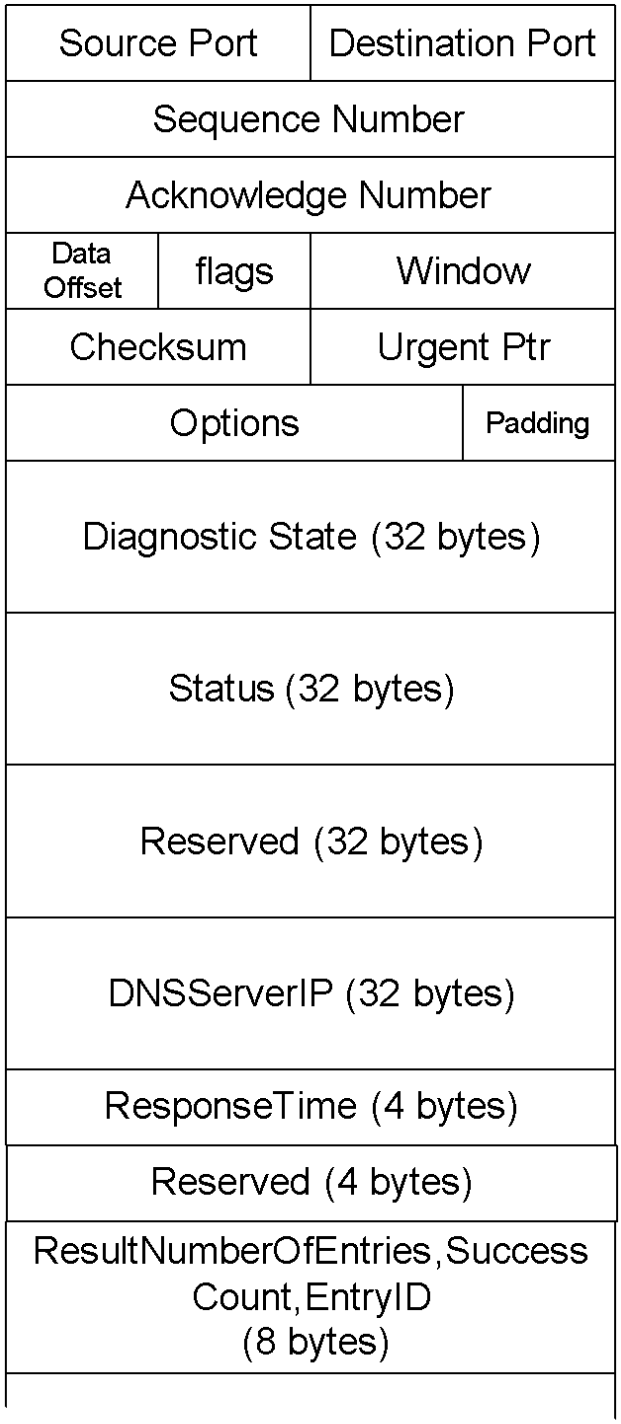

Figure 15 – TestResult packet for DNS Response Test

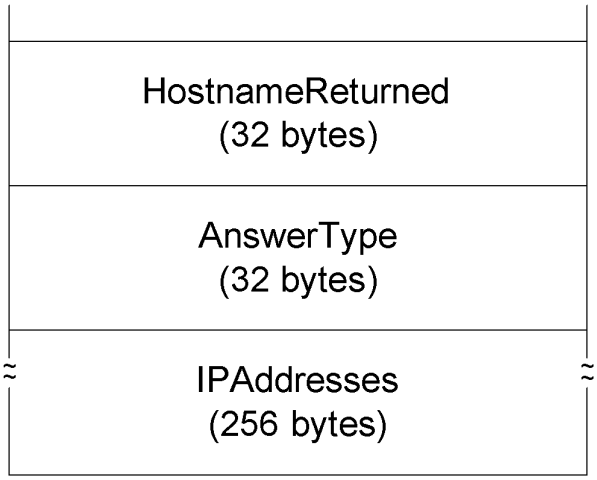

Figure 16 – TestResult packet for DNS Response Test (continued)

[C.2] A DNS test can be started by either the ACS setting the DiagnosticState variable to "Requested" or by a test server setting the PT bits to a value of 3 (PT2 = 0, PT1 = 1, PT0 = 1) in a TestResponse message while also providing the Hostname and NumberOfRepetitions strings, and setting the DTE bit to 1. A BHR can request a DNS response test be performed by setting the PT bits to a value of 3 (PT2 = 0, PT1 = 1, PT0 = 1) in a TestRequest message while also providing the Hostname and NumberOfRepetitions strings. Note that the TestRequest message from a BHR must place the Hostname wtring equivalent to the Hostname parameter in Table 5 when making the request. However, the Hostname value place in the TestResponse message by the test server is the actual Hostname the NSLookups are performed on. Also, if the Hostname field returned in the TestResponse message is an invalid Hostname, then the Hostname value in Table 5 is used instead. This way the BHR can support test servers that do not populate the Hostname field in the TestResponse messge. Once enabled NSLookups are performed by the BHR NumberOfRepetitions times and the test results are provided via a modified TestResult messages (per Figure 15 and Figure 16) for each NSLookup repetition of the test as described above. As previously mentioned, the DiagnosticState field for a DNS response time test will be pre-pended with the substring "DNS_" so a DiagnosticState value of "Completed" shall be coded as "DNS_Completed". Note that since a TestResult message is sent for each repetition of a multi-repetition DNS test, then each of NumberOfRepetitions lookups will have its own corresponding TestResult message. The DiagnosticState, ResultNumberOfEntries, and SuccessCount values returned will be identical in each of the TestResult messages (as

these are the global non-indexed values returned for the DNS test as defined in Table 5). However, the Status (e.g. “Success”, "Error_HostNameNotResolved”, etc …), DNSServerIP (e.g. “8.8.8.8”), ResponseTime, EntryID (e.g. 0, 1, 2, 3, …), HostnameReturned (e.g. www.google.com), AnswerType (e.g. “Authoritative”, “NonAuthoritative”), and IPAddresses (e.g. “20.20.1.1, 20.20.1.2”) field values will be unique to each repetition (and hence unique to each TestResults message as these values correspond to the indexed values in Table 5). The Status, DNSServerIP, HostnameReturned, AnswerType and IPAddresses are all returned as strings, the ResponseTime is returned as an unsigned integer (in milliseconds), and each of these values has a dedicated field in the TestResults message for the corresponding indexed value in Table 5. Note that the ResponseTime value is returned as a string indicating the number of milliseconds required to perform a successful lookup. The ResponseTime field is 4 bytes, so a NSLookup taking 54 milliseconds will be coded in the ResponseTime field as 00 00 00 00 36. Note that the IPAddresses field holds a string comprised of a comma separated list of all of the IP addresses returned for each NSLookup repetition (e.g. 197,132.10.1,197.132.10.2,197.132.10.3, etc …).The ResultNumberOfEntries,SucessCount,EntryID field in Figure 15 contains a comma separated string representation of the ResultNumberOfEntries, SuccessCount, and EntryID (i.e. the index) values from Table 5. Note that while the ResultNumberOfEntries and SuccessCount values do not change for each TestResult message, the EntryID value is unique each repetition (i.e. increments by 1 for each repetition). So assuming NumberOfRepetitions = 4 and SuccessCount = 3, one would see the ResultNumberOfEntries,SucessCount,EntryID field equal to “4,3,0” in the first TestResult message (Note: EntryID begins at zero), equal to “4,3,1” in the second TestResult message, equal to “4,3,2” in the third TestResult message, and equal to “4,3,3” in the fourth TestResult message. The DNS TestResult messages are then followed by a TestLocation message before closing the test results TCP connection.

Appendix D – WebPage Response Time Test Protocol

This section provides a specification for the client behavior of the WebPage response time test protocol which serves as an additional test type option. This test type is supported on devices that can leverage an existing web client implementation. The WebPage response time test utilizes the HTTP download test parameters with the definitions of some of those parameters adjusted to account for a complete web page download as opposed downloading of a single object (as is the case for the HTTP test). Thus for the web page test, the DownloadURL is assumed to represent an HTML file of a web page which is downloaded and parsed with a web client and all the objects in that HTML are subsequently downloaded and parsed. Since this is only a test, however, the contents of the HTML file and all of its objects are only downloaded to measure the response time of the complete page download and are not required to be stored or displayed. In addition to measuring the response time of a complete web page download, this test measures the total number of objects and the response times for each of the objects that comprise the page, the DNS response times for each DNS lookup performed during the page download, and a cumulative sum of the DNS response times for all the DNS lookups required to complete the download of the page and sets that sum to the DNSResponseTotal variable. As such, the DNSResponseTotal allows one to infer the proportion of the page response time that was contributed by DNS lookups.

[D.1] The WebPage response test is requested by setting the PT bits to a value of 4 (PT2 = 1, PT1 = 0, PT0 = 0) in a TestRequest packet. Regardless of what test type is requested in a TestRequest packet, the WebPage response test is to be performed if the PT bits are coded to 4 in the TestResponse packet and DTE = 1.

[D.2] We propose adding the following additional parameters for the WebPage response test at the path location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.WebPage for the purpose of incorporating it to the TR-143 test suite. VerboseWebPageDiagnostics (boolean), NumberOfWebObjects (unsigned int), NumberOfTCPConnections (unsigned int), NumberOfDNSLookups and DNSResponseTotal (unsigned int – milliseconds). VerboseWebPageDiagnostics (boolean) when true, indicates that all of the parameters in the group InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.WebPage should be computed and returned in a TestResult message. When VerboseWebPageDiagnostics is false, only the non-indexed parameters (e.g. NumberOfWebObjects, NumberOfTCPConnections, NumberOfDNSLookups DNSResponseTotal), in the InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.WebPage group are computed and returned in the TestResult message to the test server. Note that the web page response parameters in common with the HTTP throughput tests (e.g. ROMTime, BOMTime, EOMTime, TestBytesReceived, TotalBytesReceived, etc…) are also computed regardless of how VerboseWebPageDiagnostics is set. NumberOfWebObjects is the total number of web objects that comprise the web page and are downloaded as part of the web page download test. NumberOfTCPConnections is the total number of tcp

connections opened by the HTTP 1.1 or HTTP 1.0 protocol used for the web page download test. NumberOfDNSLookups is the total number of DNSL lookups performed during the web page download. DNSResponseTotal is defined as the sum of response times of all the DNS lookups contributing to a web page download.

In addition we add InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics. WebPage.WebPageResults{i} which contains the following parameters for each WebObject detected in the html page and subsequently downloaded: HttpStatus (string), TCPConnectionID (unsigned int), WebObjectURL (string), WebObjectSize (unsigned int - bytes), ResponseTime (unsigned int – milliseconds). TCPConnectionID (unsigned int) is a id for the tcp connection that the i–th web object was downloaded on, WebObjectURL is a string representation of the URL downloaded for i-th web object in the page, HttpStatus is a string representation the status code returned for the i-th web object (e.g. "200 OK", etc …), WebObjectSize is the size of that i-th web object in bytes, and ResponseTime is the amount of time in milliseconds it took to download that i-th object of the page. So a web page consisting of 250 objects would contain 250 indexed WebObjectResults{i} parameters each having its own TCPConnectionID, HttpStatus, WebObjectURL, WebObjectSize, and ResponseTime. Note that WebObjectResults{0} corresponds to the primary html page that contains the references to the remaining objects. Also we propose to add InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics. WebPage.WebPageDNSDiagnostics{j} which contain the following parameters for each NS lookup performed to complete the web page download: Status (string). Hostname (string), DNSServerIP (string), IPAddresses (string) and ResponseTime (unsigned int – milliseconds). These are the are the DNS parameters measured for the j-th NSLookup performed to complete the web page download. Status (string) represents the status of the j-th DNS lookup performed on the page (e.g. "Successful", "Error_HostNameNotResolved", etc …), Hostname (string) is the hostname the lookup is performed on, DNSServerIP (string) is the IP address for the server used to perform the DNS lookup, IPAddresses (string) is a string representing the IP address (or IP addresses) returned from the DNS lookup, and ResponseTime (unsigned int) is the response time of the DNS lookup in milliseconds. Note that the parameter DNSResponseTotal is equal to the sum of all of the ResponseTime values in InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics. WebPage.WebPageDNSDiagnostics{j}. With these new parameters added, the following Table is added to support the Web Page Response time test:

Name	Type	Write	Description	Object Default

.WebPage.	object	-	This object allows the CPE to be configured to perform the WebPage Response time test.	-
VerboseWebPageDiagnostics	boolean	w	Indicates if the indexed parameters in this group are computed and included in the TestResult message returned to the test server.	-
NumberOfWebObjects	unsignedInt	w	The total number of web objects which comprise the web page.	-
NumberOfTCPConnections	unsignedInt	w	The number of parallel tcp connections used by the HTTP protocol to perform the web page download.	-
NumberOfDNSLookups	unsignedInt	w	The number of DNS lookups performed during the web page download.	
DNSResponseTotal	unsignedInt	w	The sum of the response times of all of the DNS lookups used to download the web page	-
.WebObjectResults{i}.	object	w	This object allows the CPE to be configured to provide indexed results for each web object that comprise the downloaded web page.	-
TCPConnectionID	unsignedInt	w	An incrementing number assigned to each tcp connection used by the HTTP protocol t perform the web page download. The first tcp connection is assigned the number 0, the second tcp connection is assigned number 1, and so on.	-
HttpStatus	string	w	Http status code returned for the i-th web object (e.g. “200 OK”, etc …),	-
WebObjectURL	string	w	The URL for the i-th object of the web page	-
WebObjectSize	unsignedInt	w	The size of the i-th object of the web page in bytes	-
ResponseTime	unsignedInt	w	The response time for the i-th object of the web page in milliseconds.	
.WebPageDNSDiagnostics{j}.	object	w	This object allows the CPE to be configured to provide indexed results for each DNS lookup performed during the web page download.	-
Status	string	w	The Status returned for the i-th DNS lookup performed during the web page download.	-
Hostname	string	w	The hostname that the i-th DNS lookup is performed on during the web page download.	-
DNSServerIP	string	w	The IP address of domain name of the DNS server used for the i-th NS lookup of the web page download.	-
IPAddresses	string	w	The IP addresses returned from the i-th NS lookup performed during the web page download.	-
ResponseTime	unsignedInt	w	The response time for the i-th NS lookup performed during the web page download.	-

Table 6 – WebPage Object Table

[D.3.] Web Page Response Time Test Theory Of Operation

WebPage Response Time Test:

1. The DownloadURL is assumed to contain the URL of a web page (e.g. www.google.com). First Open the TCP connection to download the HTML file contained in DownloadURL. The timestamp of the Open request and SYN-ACK received in response to that open request are set to the TCPOpenRequestTime and TCPOpenResponse in the DownloadDiagnostics object.

2. Similarly, the GET request and HTTP server response (e.g. 200 OK), respectively, for the main page html file are set to ROMTime and BOMTime, respectively, in DownloadDiagnostics.
3. However, unlike the single object download HTTP test, a web client parses the HTML and triggers downloads for all the objects required to display the page if this were a true client. The EOMTime is not set to the time of arrival of the last data unit of the HTML object, but in this case is set to the timestamp of the arrival of the last data unit of the whole web page. Also, the total number of bytes in the whole page (i.e. comprised of all the page objects) are counted and set to TestBytesReceived. Also measure TotalBytesReceived across the interface for the duration of the test.
4. The sum of the response times of all the DNS lookups required to performed the complete page download is stored as DNSResponseTotal (in milliseconds).
5. Determine the DiagnosticState in a manner analogous to the single object HTTP test.
6. Upon completion of the WebPage response time test, we have DiagnosticState, TCPOpenRequestTime, TCPOpenResponse, ROMTime, BOMTime, EOMTime, TestBytesReceived, TotalBytesRecevieved, and DNSResponseTotal all of which, with the exception of DNSResponseTotal can be placed in a standard TestResult packet. When sending a TestResult packet to the test server pre-pend the substring "WebPage_" and append a whitespace character followed by DNSResponseTotal coded in ascii in the DiagnosticState string. For example, if the measured DNSResponseTotal is 4 seconds, then a diagnostic state of "Completed" is forwarded as "WebPage_Completed 4000" in the TestResult packet, to distinguish a WebPage response test result from a standard HTTP test result at the test server.
7. If VerboseWebPageDiagnostics = true, then the TestResult message is followed by N, "WebObjectResult" messages, which are in turn followed by M, "WebPageDNSDiagnostics" messages (where N = NumberOfWebObjects, and M = NumberOfDNSLookups). The format for the WebObjectResult messages are depicted in Figures 17 and 18 and WebPageDNSDiagnostics messages are depicted in Figure 19 and 20. Both the WebObjectResult messages and WebPageDNSDiagnostics have a byte size of 464 bytes (i.e. having an equivalent byte size to TestResult messages). A WebObjectResults message is distinguished at the server by the DiagnosticState field being populated with the string "WebObjectResult i", where i is a number representing the index of the WebObjectResult. Note that the the non-indexed parameters included in the WebObjectResult message (NumberOfWebObjects, NumberOfTCPConnections, NumberOfDNSLookups, and DNSResponseTotal) will have the same values for each of the WebObjectResult messages. The TestResult message (described in step 6 above), is followed by N = NumberOfWebObjects, WebObjectResult messages which are in turn followed by M = NumberOfDNSLookups messages. A WebPageDNSDiagnostics meesage is distinguished by at the server by the DiagnosticState field being populated with the string "WebPageDNSDiagnostics i", where i is a number representing the index of the WebPageDNSDiagnostics.

Note that “every” parameter in the fields of both the WebObjectResult messages and NumberOfDNSLookups messages are coded a string values in the messages, including the parameters which are unsinged integers in Table 6 (e.g. WebObjectSize, ResponseTime, etc …). Also note that the WebObjectResult message and WebPageDNSDiagnostics message contain two different response times. The WebObjectResult contains the i-thWebObject Response time (and other WebObjectResult parameters as depicted in Table 6), while the WebPageDNSDiagnostics i-th DNS Response time (and other WebPageDNSDiagnostics parameters as depicted in Table 6).

8.

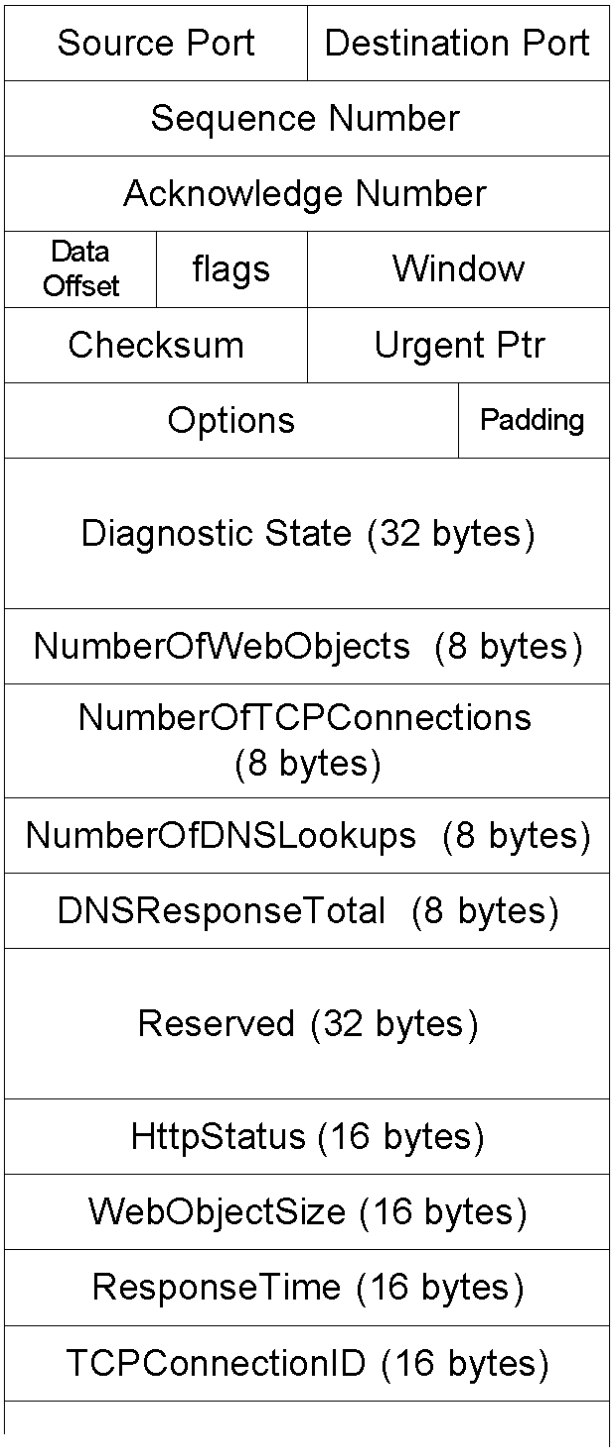

Figure 17 – WebObjectResult Message

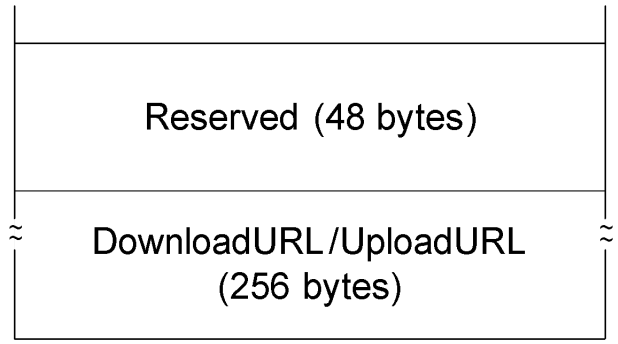

Figure 18 - WebObjectResult Message (continued)

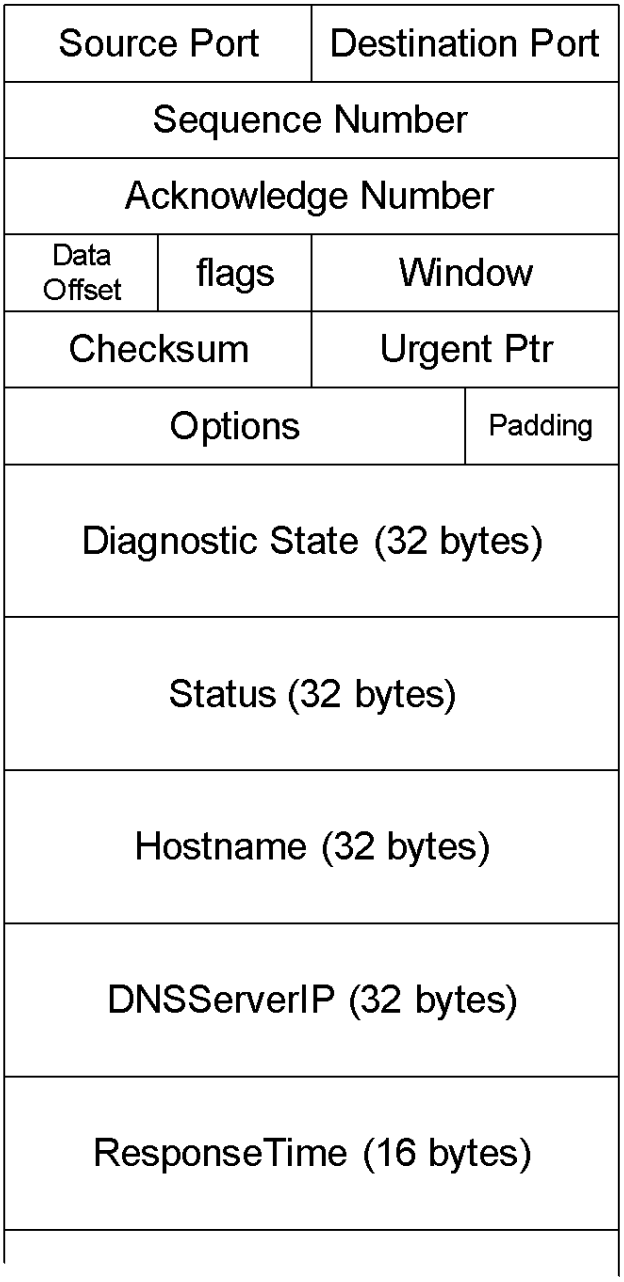

Figure 19 - WebPageDNSDiagnostics Message

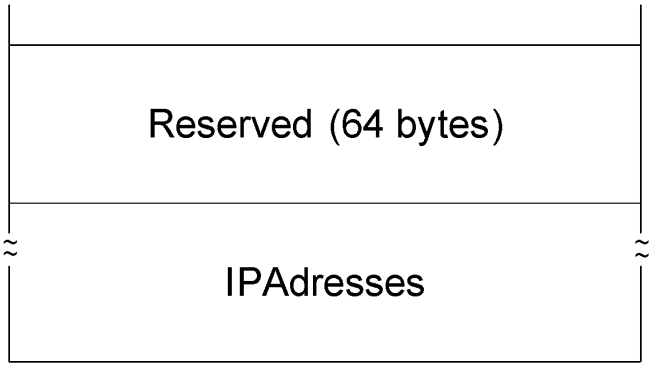

Figure 20 - WebPageDNSDiagnostics Message (continued)

Appendix E – Reverse UDP Echo Plus

This section provides a specification for the client behavior of the Reverse UDP Echo Plus test protocol. For the standard UDP Echo Plus test described in Section 2.3, the Broadband Home Router functions as the responder while the test server functions as the client. For the Reverse UDP Echo Plus, however, the Broadband Home Router functions as the client while the test server functions as the responder (i.e. UDP Echo Plus server). That is, the roles are reversed.

[C.1] We propose adding the following parameters to the UDPEchoPlus test at the path location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.UDPEchoConfig for the purpose of incorporating it to the TR-143 test suite – DiagnosticState (string), UDPEchoClientMode (boolean), UDPEchoTestInterval (unsignedInt), HostTarget (string [256]), NumberOfRepetitions (unsignedInt), TimeOut (unsignedInt), DataBlockSize (unsignedInt), DSCP (UnsignedInt), InterArrivalTime (unsignedInt – milliseconds), SuccessCount (UnsignedInt), FailureCount (unsignedInt), AverageResponseTime (unsignedInt), MinimumResponseTime (unsignedInt), MaximumResponseTime (unsignedInt). In addition we add InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.UDPEchoConfig.UDPEchoConfig.TestProbeResults{i} which contains the following parameters for each UDPEchoPlus packet sent from a BHR functioning as a UDP Echo Plus client: PacketSendTime (dateTime), PacketRcvTime (dateTime), TestGenSN (unsignedInt), TestRespSN (unsignedInt), TestRespRecvTimeStamp (unsignedInt), TestRespReplyTimeStamp (unsignedInt), TestRespReplyFailureCount (unsignedInt). With these new parameters added (shown in BOLD in the table), then Table 2 of section 2.l3 becomes:

Name	Type	Write	Description	Object Default
.UDPEchoConfig.	object	-	This object allows the CPE to be configured to perform the UDP Echo Service defined in [4] and UDP Echo Plus Service defined in Appendix A.1.	-
DiagnosticState	string	W	Indicates availability of diagnostic data. One of: “None” “Requested” “Complete” “Error_CannotResolveHostName” “Error_Internal” “Error_Other” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate the corresponding diagnostic test.	-

			When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the test is completed, the value of this parameter MUST be either Complete (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Complete, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
UDPEchoClientMode	boolean	W	Specifies whether the CPE device functions as a UDPEchoPlus client (when set to True), or a UDPEchoPlus responder (when set to False). By default, the CPE device functions as a UDPEchoPlus responder (i.e. server) so the parameter defaults to False.	-
UDPEchoTestInterval	unsignedInt (seconds)	W	A recurring fixed time interval (in seconds) within which UDPEcho (or Echo Plus) tests are performed when the CPE is in client mode (UDPEchoClientMode = TRUE). The CPE functions as a UDP Echo Plus client when EchoPlusEnabled = TRUE, and EchoPlusSupported = TRUE, and DataBlockSize is greater than or equal to 20.	
Enable	boolean	W	MUST be enabled to receive UDP echo. When enabled from a disabled state all related timestamps, statistics and UDP Echo Plus counters are cleared.	-
Interface	string(256)	W	IP-layer interface over which the CPE MUST listen and receive UDP echo requests on. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE MUST listen and receive UDP echo requests on all interfaces.	-

			Note: Interfaces behind a NAT MAY require port forwarding rules configured in the Gateway to enable receiving the UDP packets.	
SourceIPAddress	string	W	The Source IP address of the UDP echo packet. The CPE MUST only respond to a UDP echo from this source IP address.	-
UDPPort	unsignedInt	W	The UDP port on which the UDP server MUST listen and respond to UDP echo requests.	-
EchoPlusEnabled	boolean	W	If True the CPE will perform necessary packet processing for UDP Echo Plus packets.	-
EchoPlusSupported	boolean	-	True if UDP Echo Plus is supported.	-
PacketsReceived	unsignedInt	-	Incremented upon each valid UDP echo packet received.	-
PacketsResponded	unsignedInt	-	Incremented for each UDP echo response sent.	-
BytesReceived	unsignedInt	-	The number of UDP received bytes including payload and UDP header after the UDPEchoConfig is enabled.	-
BytesResponded	unsignedInt	-	The number of UDP responded bytes, including payload and UDP header sent after the UDPEchoConfig is enabled.	-
TimeFirstPacketReceived	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The time that the server receives the first UDP echo packet after the UDPEchoConfig is enabled.	-
TimeLastPacketReceived	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 The time that the server receives the most recent UDP echo packet.	-
HostTarget	string(256)	W	Host name or address of the host to perform UDPEchoPlus tests to. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
NumberOfRepetitions	unsignedInt[1:]	W	Number of repetitions of the UDPEchoPlus test to perform before reporting the results. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
Timeout	unsignedInt[1:] (milliseconds)	W	Timeout in milliseconds for the UDPEchoPlus test. That is, the amount of time to wait for the return of a USPEchoPlus packet that was sent to the HostTarget. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
DataBlockSize	unsignedInt[1:65535] (bytes)	W	Size of the data block in bytes to be sent for each UDPEchoPlus packet. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
DSCP	unsignedInt [0:63]	W	DiffServ codepoint to be used for the UDPEchoPlus test packets. By default the CPE SHOULD set this value to zero. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
InterArrivalTime	Unsignedint[1:		The interarrival time between the NumberOfRepetitions UDPEchoPlus packets sent during a given test. This parameter applies when the CPE serves as a UDPEchoPlus client. InterarrivalTime has units of milliseconds.	
SuccessCount	unsignedInt	-	Result parameter indicating the number of successful UDPEchoPlus packets (those in which a successful response was received prior to the timeout) in the most recent UDPEchoPlus test. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
FailureCount	unsignedInt	-	Result parameter indicating the number of failed UDPEchoPlus probes (those in which a successful response was not received prior to the timeout) in the most recent UDPEchoPlus test.. This parameter applies when the CPE serves as a UDPEchoPlus client.	-
AverageResponseTime	unsignedInt	-	Result parameter indicating the average response time in milliseconds over all repetitions with successful responses of the most recent UDPEchoPlus test. If there were no successful responses, this value MUST be zero. This	-

			parameter applies when the CPE serves as a UDPEchoPlus client. AverageResponseTime has units of milliseconds.	
MinimumResponseTime	unsignedInt	-	Result parameter indicating the minimum response time in milliseconds over all repetitions with successful responses of the most recent UDPEchoPlus test. If there were no successful responses, this value MUST be zero. This parameter applies when the CPE serves as a UDPEchoPlus client. MinimumResponseTime has units of milliseconds.	-
MaximumResponseTime	unsignedInt	-	Result parameter indicating the maximum response time in milliseconds over all repetitions with successful responses of the most recent UDPEchoPlus test. If there were no successful responses, this value MUST be zero. This parameter applies when the CPE serves as a UDPEchoPlus client. MaximumResponseTime has units of milliseconds.	-
.**UDPEchoConfig.IndividualPacketResults{i}**	object	-	This object provides the results from individual UDPEchoPlus test probes (i.e. packets) collected during a UDPEchoPlus test.	-
PacketSucess	boolean	-	Indicates that the i-th UDP Echo Plus packet sent was received by the client. When this value is TRUE, then all the remaining UDPEchoConfig.IndividualPacketResults{i} parameters are valid. Otherwise only the values originally set by the CPE client (e.g. PacketSendTime and TestGenSN) MAY be set to valid values.	-
PacketSendTime	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the client sent the i-th UDP Echo Plus packet of a UDP Echo Plus test.	-
PacketRcvTime	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the client received the i-th UDP Echo Plus packet of a UDP Echo Plus test, if that packet was received by the CPE client. That is, it represents the return time for a UDP Echo Plus packet and is only valid if PacketSucess is TRUE.	-
TestGenSN	unsginedint	-	The TestGenSN value set in the i-th UDPEcho Plus packet sent by the CPE client.	-
TestRespSN	unsginedint	-	The TestRespSN value returned from the UDP Echo Plus server (i.e. HostTarget) for the i-th UDP Echo Plus packet sent by the CPE client. It is only valid if PacketSucess is TRUE.	-
TestRespRecvTimeStamp	unsignedInt	-	The 32bit unsigned integer timestamp (in microseconds) that was set by the UDP Echo Plus server (i.e. HostTarget) to record the reception time of the i-th UDP Echo Plus packet sent from the CPE client. It is only valid if PacketSucess is TRUE.	-
TestRespReplyTimeStamp	unsignedInt	-	The 32bit unsigned integer timestamp (in microseconds) that was set by the UDP Echo Plus server (i.e. HostTarget) to record the server reply time of the i-th UDP Echo Plus packet sent from the CPE client. That is, the time that the server returned the UDP Echo Plus packet. It is only valid if PacketSucess is TRUE.	-
TestRespReplyFailureCount	unsignedInt	-	The count value that was set by the UDP Echo Plus server (i.e. HostTarget) to record the number of locally dropped echo response packets by the server. This count is incremented if a valid echo request packet is received at a UDP EchoPlus server but for some reason cannot be responded to (e.g. due to local buffer overflow, CPU utilization, etc...). It is only valid if PacketSucess is TRUE.	-

Table 7 (UDP Echo Plus Table)

[C.2] When UDPEchoClientMode is set to TRUE, the BHR functions as a UDPEchoPlus client if EchoPlusEnabled or EchoPlusSupported are TRUE. Alternatively, it functions as a standard UDP Echo client if EchoPlusEnabled or EchoPlusSupported are FALSE. If it functions as a standard UDPEcho client, then the traffic generation behavior is the same though all the EchoPlus specific parameters in UDPEchoConfig.IndividualPacketResults{i} (e.g. TestGenSN, TestRespSN, TestRespRecvTimeStamp, TestRespReplyTimeStamp, TestRespReplyFailure) are ignored.The non EchoPlus specific parameters in UDPEchoConfig.IndividualPacketResults{i} (e.g. PacketSucess, PacketSendTime, PacketRcvTime) apply whether the BHR is a UDPEcho client or UDPEchoPlus client. Note that the default behavior for the BHR (i.e. when Enable is TRUE and UDPEchoClientMode is FALSE) is to function as a UDP Echo or Echo Plus server (responder).

When the BHR functions as a client (UDPEchoClientMode = TRUE), a UDP Echo Plus test is comprised of NumberOfRepetitions packets sent from the BHR (i.e. UDPEchoPlus client) to the HostTarget (i.e. UDPEchoPlus server/responder), each with the TestGenSN field filled. Each UDP Echo Plus packet is in turn returned from the HostTarget with TestRespSN, TestRespRecvTimeStamp, TestRespReplyTimeStamp, and TestRespReplyFailureCount fields and TestRespReplyIncomingDiscards*, TestRespReplyOutgoingDiscards*, TestRespRecvDateTimeStr*, TestRespRecvIncomingBufferOcc* and TestRespRecvOutgoingBufferOcc* experimental fields filled by the HostTarget. That is, the UDPEchoPlus test is performed using the same packet format as specified in Figure 3.

The InterArrival time parameter is the time interval spacing between each of the UDP Echo Plus packets sent and each packet is coded to a priority level dictated by the DSCP parameter. The payload size of each UDP Echo Plus packet is dictated by the DataBlockSize parameter. UDP Echo Plus fields must be populated whenever the DataBlockSize is greater than or equal to 20 bytes. Otherwise standard UDP Echo packets are sent. When UDPEchoClientMode = TRUE, Reverse UDP Echo (or Echo Plus) tests are triggered in three ways: (1) By setting the UDPEcho DiagnosticState = “Requested” to trigger a single test, or (2) periodically once every UDPEchoTestInterval seconds when UDPEchoTestInterval is set to a value greater than zero, or (3) when the PT field is set to 5 (i.e. PT0 = 1, PT1 = 0, PT2 = 1) and DTE = 1, to denote UDP Echo Plus test in a TestResponse packet (in response to a TestRequest). The TestResponse packet based triggering enables the CPE client to test to an explicitly specified HostTarget which is also provided in the TestResponse packet (via the URL field when URL bit is set to 1). A TestResponse packet with PT = 5 takes precedence in triggering a UDP Echo Plus test so if a TestResponse with PT = 5 is received by the BHR prior to a UDPEchoTestInterval period ending, the test is performed due to the TestResponse packet and the next test is scheduled to occur UDPEchoTestInterval seconds from the completion of the UDPEchoPlus test that just occurred. For example, assume that UDPEchoTestInterval = 5 seconds and that a UDP Echo Plus test occurs at time t = 5s Then the next test will be scheduled to occur at time t = 10s. However, if the BHR sends

a TestRequest just prior to time t = 7s and receives a TestResponse suggesting that a UDP Echo Plus test should be performed (i.e. TestResponse packet with PT = 5), then the UDP Echo Plus test is performed immediately upon receiving the TestResponse. Assuming the UDP Echo Plus test completes at time t = 7s, then the test that would have occurred at time t = 10s is cancelled and the next test is scheduled to be performed at time t = 12s and every 5 s thereafter (unless another TestResponse trigger test occurs).

Note that UDPEchoTestInterval can be set to zero, in which case all of the UDPEcho (or EchoPlus) tests must be triggered from TestResponse packets (or by setting the UDPEcho DiagnosticState = "Requested") since none are scheduled at the BHR. Whether the UDP Echo Plus test is triggered from a DiagnosticState = "Requested" setting, or a UDPEchoTestInterval timer expiration or a TestResponse packet (with PT = 5 and DTE=1), all the remaining test parameters (e.g. DSCP coding, DataBlockSize, InterarrivalTime, NumberOfRepetitions, etc …) outside of the HostTarget which is provided in the TestResponse packet are values used from Table 6.

Note that the DiagnosticState parameter is applicable when UDPEchoClientMode = TRUE, and is set according to the same rules used for the ICMP Echo (ping) test as specified in TR-098 following a UDP Echo or UDPEchoPlus test.

[C.3] The TestResult message is sent to the test server via a TCP connection after the ReverseUDPEchoPlus test completes, just as is the case for throughput test results. Because the UDP Echo Plus test is not a throughput test, it requires some modifications to the TestResult message field definitions. Let us first define a "UDPEchoPlus data record" as 122 byte or 320 byte data structure. The 122 byte UDPEchoPlus data record is only retained to provide backwards compatibility with previous implementations of ReverseUDPEchoPlus. Only the 320 byte UDPEchoPlus data record should be implemented in any new implementations. The UDPEchoPlus data record comprised of the following fields:

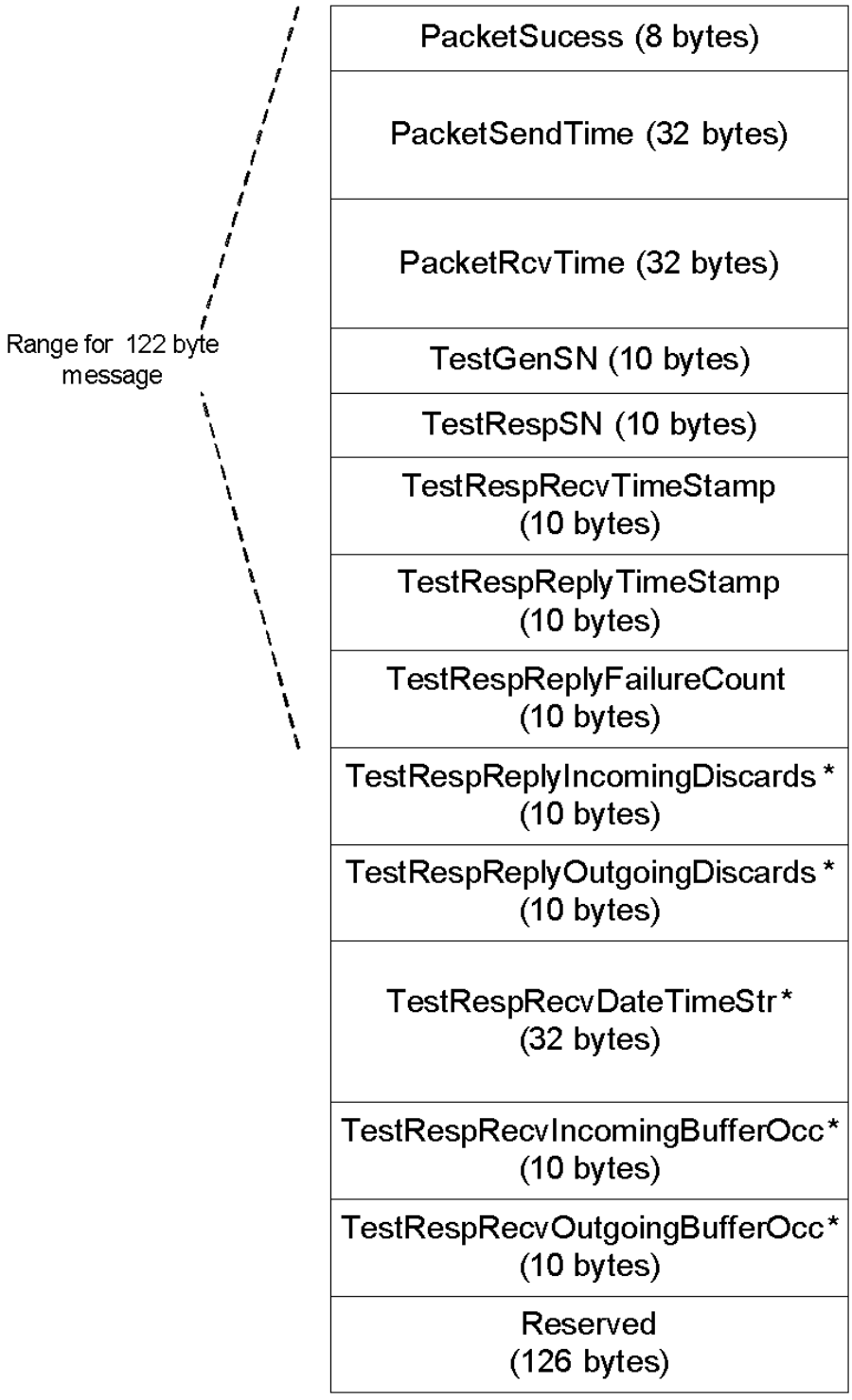

Figure 21 (UDPEchoPlus Data Record)

The 8 byte PacketSucess field, as defined in Table 7 (UDP Echo Plus Table), is set to either “TRUE” or “FALSE” (for the 122 byte data record) according to whether or not a UDP Echo or Echo Plus packet is successfully sent to and returned from HostTarget. However, to indicate the 320 byte data record is used, then the PacketSucess field is set to either “TRUE_E” or “FALSE_E”. That is, to suggest that the longer 320 byte data record is used, an underscore and the character ‘E’ is appended to the “TRUE” or “FALSE” string value (to indicate that the UDPEchoPlus extended data record is being used). All of the fields up to and including the TestRespReplyFailureCount field are included in both the 122 byte and 320 byte data record formats, and the remaining fields are only included in the 320 byte data record format.

The 32 byte PacketSendTime and PacketRcvTime fields, as defined in Table 7 (UDP Echo Plus Table), are the dateTime string values in UTC specified to microsecond precision (e.g. 2008-04-09T15:01:05.123456). These timestamps represent the times when the UDP EchoPlus packet was sent to and received from the HostTarget, respectively. Note that if the test was a standard UDP Echo test then only the PacketSucess, PacketSendTime and PacketRcvTime fields need to be correctly populated by the BHR client. For the UDPEchoPlus test the remaining TestGenSN, TestRespSN, TestRespRecvTimestamp, TestRespReplyTimestamp, and TestRespReplyFailure fields, as defined in Table 7 (UDP Echo Plus Table), are also populated for both the 122 byte and 320 byte data records (assuming PacketSucess is TRUE), and the remaining fields (TestRespReplyIncomingDiscards*, TestRespReplyOutgoingDiscards*, TestRespRecvDateTimeStr*, TestRespRecvIncomingBufferOcc*, and TestRespRecvOutgoingBufferOcc*) are populated when the 320 byte data record format is used. However, the TestRespRecvIncomingBufferOcc* and TestRespRecvOutgoingBufferOcc* are not populated with the buffer occupancy values returned from HostTarget (i.e. the test server), but are populated with buffer occupancy values measured at the BHR itself at the time the UDPEchoPlus packet is transmitted (for the TestRespRecvOutgoingBufferOcc* measurement) and received (for the the TestRespRecvIncomingBufferOcc* measurement). That is, TestRespRecvIncomingBufferOcc* and TestRespRecvOutgoingBufferOcc* are measured by the BHR during the ReverseUDPEchoPlus test and the BHR measuremnts are placed in the UDPEchoPlus data record instead of the buffere measurement values returned from HostTarget per UDPEchoPlus packet as depicted in Figure 3 (UDP Echo Plus Packet Format).

Note that a data record corresponds to a single UDP Echo or UDP Echo Plus packet (per Figure 3 (UDP Echo Plus Packet Format)) sent to and returned from the HostTarget during the UDP Echo or Echo Plus test. However, each field in the UDP Echo Plus data record is populated with a string representation of each of the values for TestGenSN, TestRespSN, TestRespRecvTimeStamp, TestRespReplyTimeStamp, TestRespReplyFailureCount, TestRespReplyIncomingDiscards*, TestRespReplyOutgoingDiscards*, TestRespRecvDateTimeStr*, TestRespRecvIncomingBufferOcc*, and TestRespRecvOutgoingBufferOcc* though these paramters are actually unsigned integer values as depicted in Figure 3 (UDP Echo

Plus Packet Format) and Table 7 (UDP Echo Plus Table), with the exception of TestRespRecvDateTimeStr* which is already a string .

So to summarize, the PacketSucess field is set to either “TRUE” or “FALSE” to indicate the 122 byte data record, or “TRUE_E” or “FALSE_E” to indicate the 320 byte data record. The 32 bytes dataTime fields (PacketSendTime and PacketRcvTime) are already string formatted as defined in Table 7 (UDP Echo Plus Table), and the remaining fields (TestGenSN, TestRespSN, etc …), with the exception TestRespRecvDateTimeStr*, are unsigned integers, as defined in Table 7 (UDP Echo Plus Table), but are populated as string representations in the UDP Echo Plus data record.

When the UDP Echo or UDP Echo Plus client mode is enabled (i.e. UDPEchoClientMode = TRUE), then the results of the UDP Echo Plus test are passed to the test server using a TestResults TCP connection over TestResultsTCPPort to TestServerIPAddress upon completion of the UDP Echo or Echo Plus test. The format the UDP Echo or Echo Plus test results message is:

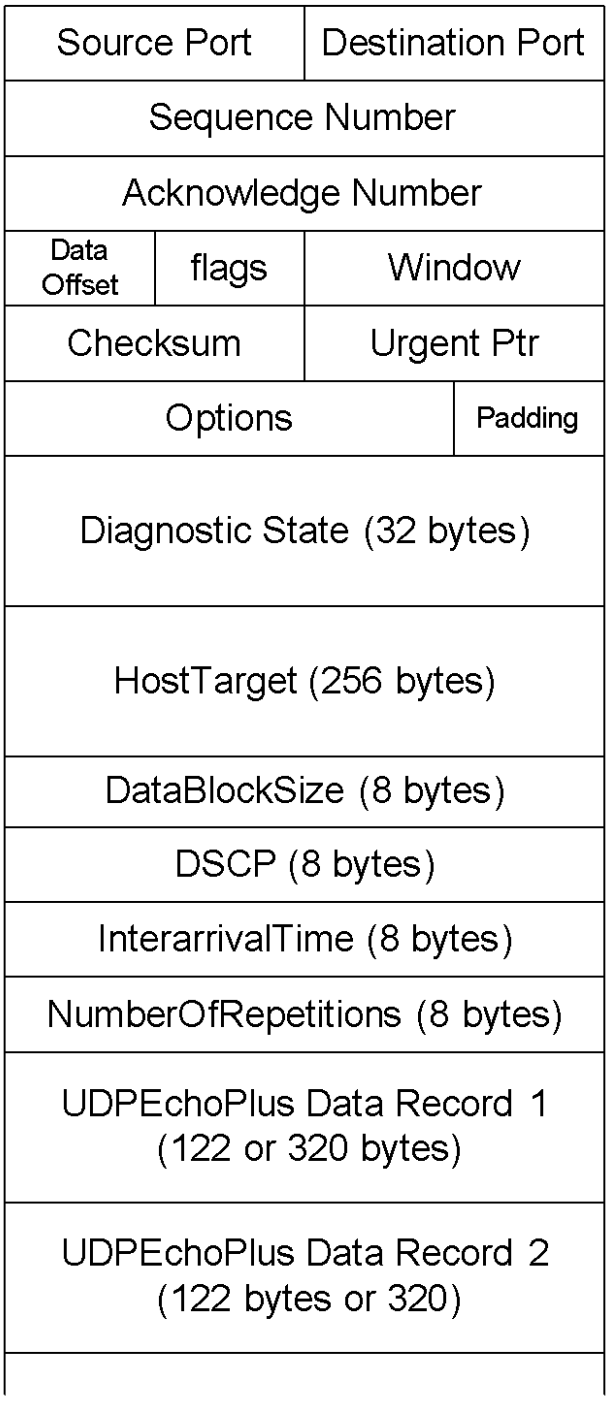

Figure 22 (UDPEchoPlus TestResult message Part 1)

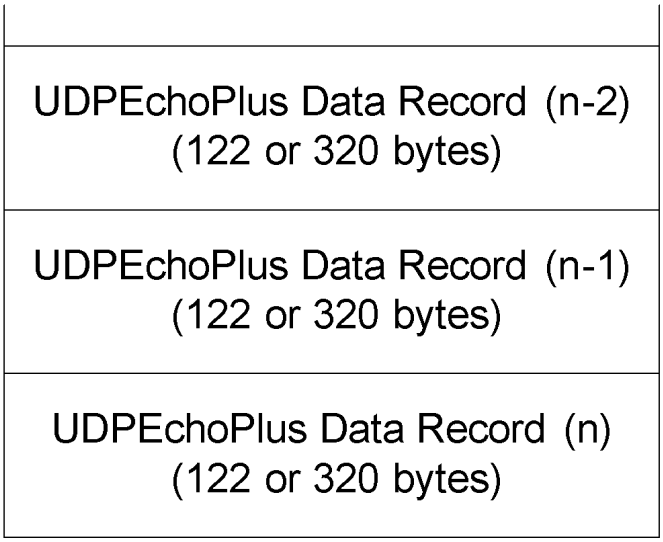

Figure 23 (UDP Echo Plus TestResult message Part 2)

The total number of data records to be transferred is n = NumberOfRepetitions (as specified in the NumberOfRepetitions field). Note that the DiagnosticState field is the same string value as defined in Table 7 (UDP Echo Plus Table), but with the substring "UDPEcho_" pre-pended. So a successful test that would result in the DiagnosticState parameter in Table 7 (UDP Echo Plus Table) being set to "Complete", but will be conveyed with the DiagnosticState field of the UDPEchoPlus TestResult packet set to "UDPEcho_Complete", and so on. This way, the test server can decipher that the test result is a UDP Echo or Echo Plus test result and in turn parse the remainder of the TestResult message correctly. The remaining four fields, DataBlockSize (8 bytes), DSCP (8 bytes), InterArrivalTime (8 bytes), and NumberOfRepititions (8 bytes), are all formatted as binary representations of unsigned integers (e.g. DataBlockSize = 31 is passed in the DataBlockSize field as 000000000000001f). Note that the header fields of the UDP Echo Plus TestResult message (i.e. all the fields prior to the data records) require 320 bytes so there is room in a single 1460 byte tcp segment to transfer about nine 122 byte UDPEchoPlus data records or three 320 byte data records, and about eleven 122 byte data records or four 320 byte data records in each tcp segment thereafter.

Appendix F – Two Way Active Monitoring Protocol (TWAMP)

This section provides a specification for the server and client behaviors of the Two Way Active Monitoring Protocol (TWAMP) protocol as defined in RFC 5357. In particular, it specifies the functions the device should support in the RFC and provides a Table of parameters for configuration and results collection. By server function we mean the device functions as a responder (or what is referred to in the RFC as the control server and Session-Reflector functions in the two host model). For the standard UDP Echo Plus test described in Section 2.3, the Broadband Home Router functions as the responder while the test server functions as the client. For the Reverse UDP Echo Plus, however, the Broadband Home Router (BHR) functions as the client while the test server functions as the responder (i.e. UDP Echo Plus server). That is, the roles are reversed. The same applies to TWAMP. The device can function either as a client (when in TWAMP client mode) or as a server/responder (when in TWAMP server mode). In addition, the BHR can function as a lite server (when in TWAMP lite-server mode) as defined in Appendix I of RFC 5357. Note that when in TWAMP client mode, the BHR functions as both a control-Client and Session Sender, when in TWAMP server mode, the BHR functions as both a control server and Session-Reflector, and when in TWAMP lite-server mode the BHR functions as just a Session-Reflector.

[F.1] We propose adding the following TWAMP test parameters to the device at the path location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.TWAMPConfig for the purpose of incorporating it to the TR-143 test suite – DiagnosticState (string), TWAMPClientMode (boolean), TWAMPLite (boolean), TWAMPTestInterval (unsignedInt), TWAMPHostTarget (string [256]), TWAMPNumberOfRepetitions (unsignedInt), TWAMPTimeOut (unsignedInt), TWAMPDataBlockSize (unsignedInt), TWAMPDSCP (UnsignedInt), TWAMPInterArrivalTime (unsignedInt – milliseconds), TWAMPSuccessCount (UnsignedInt), TWAMPFailureCount (unsignedInt), TWAMPAverageResponseTime (unsignedInt), TWAMPMinimumResponseTime (unsignedInt), TWAMPMaximumResponseTime (unsignedInt), TWAMPServWait (unsignedInt – seconds), TWAMPPassPhrase (string[256]), TWAMPTestPort (unsignedInt), TWAMPRefWait (unsignedInt – seconds), TWAMPStopTestTimeout (unsignedInt – seconds), TWAMPEncryptionMode (string[32] – “Unauthenticated”, “Authenticated”, or “Encrypted”), TWAMPPaddingMode (boolean). In addition we add InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.TWAMPConfig.IndividualPacketResults{i} which contains the following parameters for each TWAMP test packet sent from a BHR functioning as a TWAMP client: SenderSequenceNumber (unsignedInt), SenderTimestamp (dateTime),

SenderErrorEstimate (unsignedInt), SenderTTL (unsignedInt), SequenceNumber (unsignedInt), Timestamp (dateTime), ErrorEstimate (unsignedInt), ReceiveTimestamp (dateTime). With these new parameters added then we have the following Table:

Name	Type	Write	Description	Object Default
.TWAMPConfig.	object	-	This object allows the CPE to be configured to perform the TWAMP test as defined in RFC 5357. The device functions as a control-client and Session-Sender, or as a control-server and Session-Reflector, or as a Session-Reflector in the TWAMP lite model (defined in Appendix I of RFC 5357).	-
DiagnosticState	string	W	Indicates availability of diagnostic data when the device functions as a TWAMP client. The value may be one of: “None” “Requested” “Complete” “EncryptionModeNotSupported” “Error_CannotResolveHostName” “Error_Internal” “Error_Other” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate a TWAMP test every TWAMPTestInterval seconds for TWAMPTestCount interations. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the each test is completed, the value of this parameter MUST be either Complete (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Complete, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result	-

			in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	IP-layer interface over which the CPE MUST listen and receive TWAMP control and test session messages on. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE MUST listen and receive TWAMP control and test session messages on all interfaces. Note: Interfaces behind a NAT MAY require port forwarding rules configured in the Gateway to enable receiving the TWAMP packets.	-
TWAMPEnable	boolean	W	MUST be enabled for TWAMP logic to function in the device. When enabled from a disabled state all related timestamps, sta tistics and TWAMP counters are cleared.	-
TWAMPPassPhrase	String[256]	W	Passphrase used to TWAMP authenticated and Ecnrypted modes.	-
TWAMPTestPort	unsigedInt	W	The UDP port used for the TWAMP test session.	
TWAMPClientMode	boolean	W	Specifies whether the CPE device functions as a TWAMP rol-conclient and Session Sender (when set to True), or a TWAMP server and Session-Reflector (when set to False). By default, the CPE device functions as a TWAMP control-server and Session-Reflector so the parameter defaults to False. Note that for this parameter take effect, then TWAM PLite must be set to False.	-
TWAMPLite	boolean	W	Specifies whether the device functions as a SessionReflector. Takes priority over TWAMPClientMode parameter so when TWAMPLIte is True, the device is only a SessionReflector for TWAMPLite.	-
TWAMPEncryptionMode	String[32]		Can be set to “NonAuthenticated” (default), “Authenticated”, or “Encrypted”, to specify the authentication or encryrption model (per RFC 5357) used for the TWAMP test. If to this parameter is setany value outside of one of these strings the NonAuthenitcated mode is used.	-
TWAMPServWait	unsignedInt (seconds)	W	The amount of time the TWAMP control connection is allowed to be idle before it is closed by the control-server.	-
TWAMPRefWait	unsignedInt (seconds)	W	The amount of time the TWAMP Session-Reflector waits for some activity in the session when no packet has been received from the Session-Sender for a while (i.e. the test is idle), before assuming the test has ended.	-
TWAMPStopTestTimeOut	unsignedInt (seconds)	W	The amount of time after a Stop-Session message (per RFC 5357), that a Session-Reflector will continue to respond to (i.e. reflect) test packets received from the Session-Sender.	-
TWAMPTestInterval	unsignedInt (seconds)	W	A recurring fixed time interval (in seconds) within which TWAMP tests are performed when the CPE is in client mode (TWAMPClientMode = TRUE and TWAMPLite = False).	-
TWAMPControlServer	string(256)		The hostname or address of the host that the TWAMP control connection is established too when the CPE is functioning as a TWAMP client. Note that when this device functions as a TWAMP client, the Control Server and Session-Reflector can be different hostnames (or IP	

			addresses). When CPE functions as a TWAMP client, and the Control Server and Session Reflector are distinct, then TWAMPControlServer serves as the Control Server hostname (or IP address), and TWAMPHostTarget servers as the Session Reflector hostname (or IP address).	
TWAMPHostTarget	string(256)	W	Host name or address of the host to perform TWAMP tests to. The IP address for this parameter is placed in the Receiver Address field in the TWAMP control packets when the CPE serves as a TWAMP client. When CPE functions as a TWAMP client, and the Control Server and Session Reflector are distinct, then TWAMPControlServer serves as the Control Server hostname (or IP address), and TWAMPHostTarget servers as the Session Reflector hostname (or IP address).	-
TWAMPNumberOfRepetitions	unsignedInt	W	Number of repetitions of the TWAMP test during the TWAMP test sesssion. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPTimeout	unsignedInt (milliseconds)	W	Timeout in milliseconds for TWAMP test. That is, the amount of time to wait before a TWAMP packet sent from the SessionSender returns to the Session-Sender is determined to be lost packet. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPDataBlockSize	unsignedInt[1:65535] (bytes)	W	Size of the padding bytes to be appended to each TWAMP packet. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPPaddingMode	boolean	W	When set to True than pseudo-random data is inserted into padding bytes (if any). Otherwise, when set to False, all zeros are placed into the padding bytes (i.e. padding bytes are set to all zeros when this parameter is set to False). This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPDSCP	unsignedInt [0:63]	W	DiffServ codepoint to be used for the TWAMP test packets. By default the CPE SHOULD set this value to zero. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPInterarrivalTime	unsignedInt (milliseconds)	W	Size of the padding bytes to be appended to each TWAMP packet. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPSuccessCount	unsignedInt		When set to True than pseudo-random data is inserted into padding bytes (if any). Otherwise all zeros are placed into the padding bytes (i.e. set padding bytes to all zeros when set to false). This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPFailureCount	unsignedInt		DiffServ codepoint to be used for the TWAMP test packets. By default the CPE SHOULD set this value to zero. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPAverageResponseTime	unsignedInt (milliseconds)		The average measured round trip time in milliseconds for TWAMP packets that were sent from and returned to the Session-Sender during a TWAMP test. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPMinimumResponseTime	unsignedInt		The minimum measured round trip time in milliseconds for TWAMP packets that were sent from and returned to the Session-Sender during a TWAMP test. This parameter applies when the CPE serves as a TWAMP client.	-
TWAMPMaximumResponseTime	unsignedInt		The maximum measured round trip time in milliseconds for TWAMP packets that were sent from and returned to the Session-Sender during a TWAMP test. This parameter applies when the CPE serves as a TWAMP client.	-
.**TWAMPConfig.IndividualPacketResults{i}**	object	-	This object provides the results from individual TWAMP test probes (i.e. packets) collected during a TWAMP test.	-
PacketSucess	boolean		Indicates that the i-th TWAMP packet sent was received by the client. When this value is TRUE, then all the remaining	-

			TWAMP.IndividualPacketResults{i} parameters are valid. Otherwise only the values originally set by the CPE client (e.g. PacketSendTime, SendSequenceNumber, etc …) MAY be set to valid values.	
PacketSendTime	dateTime		Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the client sent the i-th TWAMP packet of a TWAMP test. Note that this value is equivalent to the value placed in the SendTimestamp field of the TWAMP packet though the format is different (i.e. the format in the packet is specified as 32 bit integer part and 32 bit fractional part while the format here is UTC dateTime string). The SendTimestamp is initially placed into the TWAMP packet by the Session-Sender and reflected by the Session-Reflector.	-
PacketRcvTime	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the client received the i-th TWAMP packet sent of a TWAMP test, if that packet was received by the CPE client. That is, it represents the return time for i-th TWAMP packet that was sent and is only valid if PacketSucess is TRUE.	-
SenderErrorEstimate	unsignedInt		The error estimate value which is placed into the TWAMP SenderErrorEstimate field of this TWAMP packet. The value is initially placed into the TWAMP packet by the Session-Sender (in relation to the SenderTimestamp value) and reflected by the Session-Reflector.	-
SenderSequenceNumber	unsignedInt		The sequence number value which is placed into the TWAMP SendSequenceNumber packet field of this TWAMP packet.The value is initially placed into the TWAMP packet by the Session-Sender and reflected by the Session-Reflector.	-
SenderTTL	unsignedInt		The TTL value which is placed into the TWAMP SenderTTL field of this TWAMP packet. The value is initially placed into the TWAMP packet by the Session-Sender and the value received by the Session-Reflector is returned by the Session-Reflector to the Session-Sender.	-
SequenceNumber	unsignedInt		The sequence number value which is placed into the TWAMP SequenceNumber field of this TWAMP packet. The value is placed into the TWAMP packet by the Session-Reflector.	-
Timestamp	dateTime		Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the Session-Reflector reflected the i-th TWAMP packet of a TWAMP test. Note that this value is equivalent to the value placed in the Timestamp field of the TWAMP packet though the format is different (i.e. the format in the packet is specified as 32 bit integer part and 32 bit fractional part while the format here is UTC dateTime string). The value is placed into the TWAMP packet by the Session-Reflector.	-
ErrorEstimate	unsignedInt		The error estimate value which is placed into the TWAMP ErrorEstimate field of this TWAMP packet. The value is placed into the TWAMP packet by the Session-Reflector in relation to the Timestamp value.	-
ReceiveTimestamp	dateTime		Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456,	

			The dateTime that the Session-Reflector received the i-th TWAMP packet of a TWAMP test. Note that this value is equivalent to the value placed in the ReceiveTimestamp field of the TWAMP packet though the format is different (i.e. the format in the packet is specified as 32 bit integer part and 32 bit fractional part while the format here is UTC dateTime string). The value is placed into the TWAMP packet by the Session-Reflector.	

Table 8 (TWAMP Table)

[F.2] When the BHR is functioning as a TWAMP client (i.e. when TWAMPClientMode = True). A TestResult message is sent to the test server via a TCP connection after a TWAMP test completes, just as is the case for throughput test results. Because, like the ReverseUDPEchoPlus test, the TWAMP test is not a throughput test, it requires some modifications to the TestResult message field definitions. Let us first define a "TWAMP data record" as a 320 byte data structure comprised of the following fields:

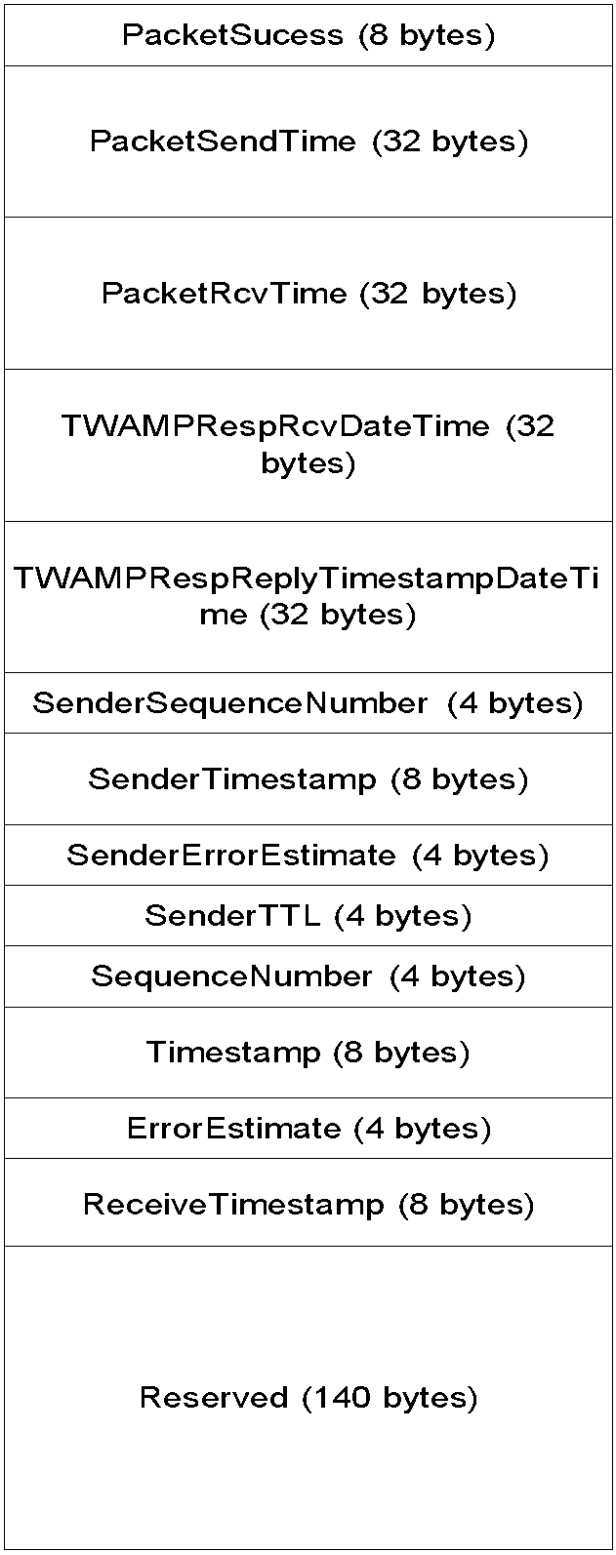

Figure 24 (TWAMP Data Record)

The 8 byte PacketSucess field (as defined in Table 8 (TWAMP Table)) is set to either the string value “TRUE” or “FALSE” according to whether or not a TWAMP packet is successfully sent to and returned from HostTarget. The 32 byte PacketSendTime and PacketRcvTime fields (as defined in Table 8 (TWAMP Table)) are dateTime string values in UTC specified to microsecond precision (e.g. 2008-04-09T15:01:05.123456). These timestamps represent the times when the TWAMP packet was sent to and received from the HostTarget, respectively. Note that PacketSendTime is equivalent to the SenderTimestamp value though they are coded in different formats. While PacketSendTime is coded as a dateTime string of the UTC value, the SenderTimestamp is specified by two unsignedInt values (the first 32 bits representing the integer value of the sender timestamp and the second 32 bits representing the fractional value of the sender timestamp. Thus the SenderTimestamp field is encoded in the same format as it is coded in the TWAMP packet. Similarly, SenderSequenceNumber, SenderErrorEstimate, SenderTTL, SequenceNumber, Timestamp, ErrorEstimate, and ReceiveTimestamp are each coded as unsigned integer values as they are in TWAMP packet itself. TWAMPRespRcvDateTime and TWAMPRespReplyDateTime are both dataTime string values in UTC specified to microsecond precision that represent the times the Session-Reflector received and subsequently transmitted the TWAMP packet. Thus they are dataTime string representations of ReceiverTimestamp and Timestamp, resepectively. Note that a TWAMP data record corresponds to a single TWAMP packet (per RFC 5357) sent to and returned from the HostTarget during the TWAMP test so a TWAMP data record is included in the TWAMP test result message for each TWAMP packet sent. If a TWAMP packet sent from the Session-Sender is lost due to a packet loss event, then PacketSucess is set to “FALSE” and the values for the SequenceNumber, Timestamp, ErrorEstimate, ReceiveTimestamp, PacketRcvTime, TWAMPRespRecvTimestamp, and TWAMPRespReplyTimestamp fields are set to all zeroes.

When the TWAMP client mode is enabled (i.e. TWAMPClientMode = TRUE), then the results of each TWAMP test are passed to the test server using a TestResults TCP connection over TestResultsTCPPort to TestServerIPAddress upon completion of the TWAMP test. The format of the TWAMP test results message is:

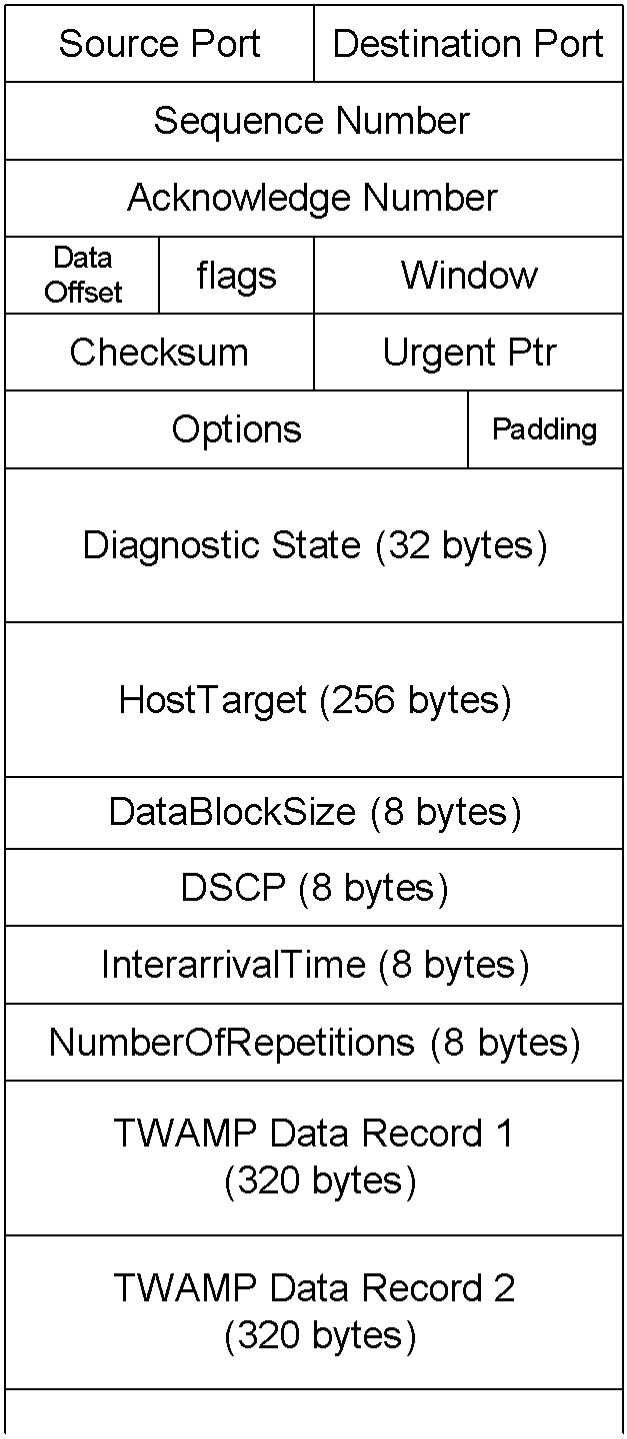

Figure 25 (TWAMP TestResult Packet)

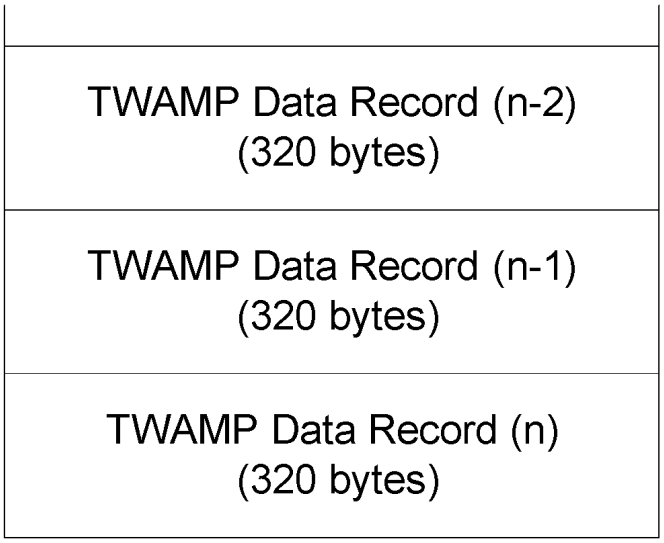

Figure 26 (TWAMP TestResult Packet – continued)

The first 320 bytes of the TWAMP TestResult packet represent the TWAMP message header including the DiagnostcState (32 byte - string), HostTarget (256 byte – string), DataBlockSize (8 byte – unsignedInt), DSCP (8 byte – unsignedInt), InterarrivalTime (8 bytes – unsignedInt), and NumberOfRepititions (8 byte – unsignedInt). This 320 byte TWAMP TestResult packet header is followed by n = NumberOfRepetition, TWAMP data records (each per Figure 24 (TWAMP Data Record)), where each TWAMP data record is 320 bytes long.

[F.3] **Theory of Operation**: The BHR will implement 3 different modes of operation of the TWAMP protocol. Per RFC 5357, a TWAMP test is composed of both a control session (over well defined TCP port 862) to configure the test, and a test session where test packets are exchanged for the purpose of measuring two-way performance metrics. The TWAMP control session is initiated by the control-client opening a TCP connection to the control-server (referred to as the "server" in RFC 5357). Once the test parameters are exchanged a test is initated using a Start-Session message. Subsequently, a test session is performed over UDP transport between a Session-Sender (which sends TWAMP test packets), and Session-Reflector which receives, timestamps, and returns the TWAMP test packets back to the Session-Sender. For the purpose of this document we assume the two host model defined in RFC 5357. When in TWAMP client mode (i.e. TWAMPClientMode = TRUE), the BHR functions as both a control-client (for the TCP control channel), and a Session-Sender (for the test session over UDP). When in TWAMP server mode (i.e. TWAMPClientMode = FALSE), then the BHR functions as both a control-server (for the TWAMP TCP control channel) and a Session-Reflector.

For the third and final mode, the BHR can function as a TWAMPLite server (i.e. when TWAMPClientMode = FALSE and TWAMPLite = TRUE), then the BHR functions solely as a Session-Reflector and assumes no TWAMP control setup is performed or required and hence the TWAMPLite server is open to receive and reflect TWAMP test packets arriving on UDP port TWAMPTestPort from any source. Note that for the device to function solely, as a Session-Reflector for TWAMP lite mode, then both TWAMPClientMode must be FALSE and TWAMPLite must be set to TRUE. So the TWAMPClientMode parameter takes precedence over the TWAMPLite parameter.

When in TWAMP client mode (TWAMPClientMode = TRUE), the BHR will open a TCP connection to TWAMPControlServer (see Table 8 (TWAMP Table)) over TCP port 862. That is, the TWAMPControlServer parameter contains the control-server hostname or IP address. Once the control-server responds with its greeting message, the BHR determines if the TWAMPEncryptionMode (see Table 8 (TWAMP Table)) value is supported by the control-server. If it is not, then the test is terminanted and DiagnosticState is set to "EncryptionModeNotSupported". If the encryption mode is supported by the control-server, then the BHR indicates to the control-server which encryption mode will be used (i.e. the value of TWAMPEncryptionMode) and the TWAMP control session proceeds. The BHR requests a TWAMP test session and the server responds with its acceptance and supporting information. Only a single test session is requested by the BHR for each BHR TWAMP test (though the TWAMP protocol actually supports the ability for a client to request multiple TWAMP tests per control session). The BHR then initiates the test by sending a Start-Sessions message to the control-server. Following the Start-Sessions message and acknowledgement from the control server, the TWAMP test session is performed between the BHR (functioning as the Session-Sender) and TWAMPHostTarget (i.e. the Session-Reflector). The IP address for the Session-Reflector is determined from TWAMPHostTarget either directly (if a TWAMPHostTarget is set to an IP address value) or via DNS lookup (if TWAMPHostTarget is set to a hostname). Finally the test session is conluded once the BHR (functioning as a control-client) sends the Stop-Sessions message to TWAMPControlServer.

When the BHR is functioning as a "server" (i.e. Control Server and Session-Reflector), the roles are reversed. When the BHR is functioning solely as a Session-Reflector, either the control session is performed between two other distinct hosts or no control session is performed at all, but the BHR will parse and reflect TWAMP test packets sent to it on UDP port TWAMPTestPort from any host (i.e, will function as a promiscuous Session-Reflector) with the test session assumed to be persistent until TWAMPLite is set to FALSE or TWAMPControlClient is set to TRUE.

Appendix G – PathPing

This section provides a specification for the server and client behaviors of the PathPing test. PathPing is a diagnostic network utility that combines the functionality of traceroute with that of ping. It provides details of the path between two hosts via traceroute, and per hop ping statistics computed from pinging each node in that path. The duration of the PathPing test is dependent on how many nodes are between the source and destination host, the number of packets used per traceroute and ping tests, and the range of packet sizes used for the ping tests. Note that the term "ping" is use more generally here to refer to an echo reponse/reply test between the BHR and a particular host in the path. Hence the ping component of the PathPing test in fact could be performed by UDPEcho, TCPEcho (using TCP SYN packets replied to with TCP SYN ACK packets), UDPEchoPlus, or standard ICMP Echo ping tests. The BHR will implement both the clent and server behaviors for this test.

[G.1] We propose adding the following PathPing test parameters to the device at the device path location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.PathPingConfig for the purpose of incorporating it to the TR-143 test suite – DiagnosticState (string), Interface (string[256]), DSCP (unsignedInt[0:63]), PathPingEnable (boolean), PathPingClientMode (boolean), PathPingTestInterval (unsignedInt), PathPingHostTarget (string[256]), PathPingTracerouteOnly (boolean), PathPingTracerouteTransport (unsignedInt[0:2] – where 0 -> ICMP, 1 -> UDP, 2 -> TCP), PathPingTraceroutePort (unsigned) , PathPingTraceroutePacketSize (unsginedInt), PathPingTracerouteQueriesPerHop (unsignedInt), PathPingTracerouteResolveHostnames (boolean), PathPingTracerouteMaxHops (unsignedInt – default 30), PathPingTimeout (unsignedInt – milliseconds), PathPingPort (unsignedInt), PathPingDestinationHopTransport (unsignedInt[0,3] - where 0 -> ICMP, 1 -> UDP, 2 -> TCP, and 3 -> UDPEchoPlus), PathPingIntermediateHopTransport (unsignedInt[0:2] – where 0 -> ICMP, 1 -> UDP, 2 -> TCP), PathPingNumberOfGroupRepetitions (unsigned), PathPingNumberOfPacketsPerGroup (unsignedInt), PathPingIntraGroupIntervarrivalTime (unsignedInt - milliseconds), PathPingInterGroupInterarrivalTime (unsignedInt - milliseconds), PathPingMinPacketSize (unsignedInt), PathPingMaxPacketSize (unsignedInt), PathPingPacketSizeStepIncrement (unsignedInt).

In addition we add InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.PathPingConfig.IndividualPacketResults{i} which contains the following parameters for each PathPing test packet (whether traceoute or ping/echo) sent from a BHR functioning as a PathPing client: PacketSucess (boolean), PacketSendTime (dateTime), PacketRcvTime (dateTime), TestGenSN (unsignedInt), TetRespSN (unsignedInt), TestRespRecvTimeStamp (unsignedInt - microseconds), TestRespReplyTimeStamp (unsignedInt - microseconds), TestRespReplyFailureCount,

Type (string), PacketSize (unsignedInt), Hostname (string). The PathPing parameters are illustrated in the following Table.

Name	Type	Write	Description	Object Default
.PathPingConfig	object	-	This object allows the CPE to be configured to perform the PathPing test. The device functions either as a client (when PathPingEnable = TRUE and PathPingClientMode = TRUE, or a server (i.e. traceroute and ping reflector) when PathPingEnable = TRUE and PathPingClientMode = FALSE).	-
DiagnosticState	string	W	Indicates availability of diagnostic data when the device functions as a PathPing client. The value may be one of: “None” “Requested” “Complete” “Error_CannotResolveHostName” “Error_Internal” “Error_Other” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate a PathPing test every PathPingTestInterval seconds unless PathPingTestInterval = 0 or is not set, in which case only a single PathPing test is run. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the each test is completed, the value of this parameter MUST be either Complete (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Complete, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to “None”.	-

			While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	IP-layer interface over which the CPE MUST listen and receive PathPing (i.e. traceroute and subsequent ping) messages on. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE MUST listen and receive PathPing messages on all interfaces. Note: Interfaces behind a NAT MAY require port forwarding rules configured in the Gateway to enable receiving the PathPing packets.	-
DSCP	unsignedInt [0:63]	W	DiffServ codepoint to be used for the PathPing test packets. By default the CPE SHOULD set this value to zero. This parameter applies when the CPE serves as a PathPing client.	-
PathPingEnable	boolean	W	MUST be enabled for PathPing logic to function in the device. When enabled from a disabled state all related timestamps, statistics and PathPing counters are cleared.	-
PathPingClientMode	boolean	W	When set to TRUE (and PathPingEnable = TRUE), the device operates as a PathPing client. When set to FALSE (and PathPingEnable = TRUE), the device funtions as a PathPing server (i.e. a traceroute server and an ICMP Ping server or UDPEcho server or TCP echo server or UDPEchoPlus server depending on the PathPingTransport setting).	-
PathPingTestInterval	unsigned (seconds)	W	A recurring fixed time interval (in seconds) within which PathPing tests are performed when the CPE is in client mode (PathPingClientMode = TRUE and PathPingEnable = TRUE).	
PathPingHostTarget	string(256)	W	Host name or address of the host to perform PathPing tests to. When the PathPing test is performed, first a traceroute is performed to PathPingHostTarget and then, for each packet size iteration, pings are performed to each intermediate host along the path observed from the traceroute followed by a set of pings performed to PathPingHostTarget.	-
PathPingTracerouteOnly	boolean	W	This parameter indicates whether the CPE should only perform the traceroute component of the PathPing test (and not the subsequent pings to each of the respective hops identified by the traceroute test). Thus when PathPingTracerouteOnly = TRUE, the PathPing test reduces to just a traceroute test. The default value for this parameter FALSE.	-
PathPingTracerouteTransport	unsigned[0:2]	W	The transport protocol used for the traceroute portion of the pathping test. Must be set to an unsigned integer in the range from 0 to 2. When set to 0 then ICMP is used. When set to 1, then UDP is used (with TTL incrementing from 1 to N for the N-hops along the path from this device to PathPingHostTarget). When set to 2, then TCP is used (with TTL incrementing from 1 to N for the N-hops along the path from this device to PathPingHostTarget).	-
PathPingTraceroutePort	unsigned	W	The UDP or TCP port to use when TCP or UDP is used as the transport protocol for the traceroute portion of a PathPing test. If this parameter is unassigned then the CPE selects the port to use for the traceroute component of the PathPing test when UDP or TCP is used as the	-

			PathPingTracerouteTransport protocol. This parameter applies when the CPE serves as a PathPing client.	
PathPingTraceroutePacketSize	unsigned (bytes)	W	The fixed packet size to use during the traceroute phase of the PathPing test. Note that while the “ping” phase of the PathPing test uses a range a packet sizes (and a bursty traffic pattern for each packet size in the range), the traceroute phase uses a single packet size and fixed number of queries to each hop. If this parameter is unassigned, it value defaults to 64 bytes. This parameter applies when the CPE serves as a PathPing client.	
PathPingTracerouteQueriesPerHop	unsigned	W	The number of traceroute packets to use for each hop during the traceroutephase of the PathPing test (which is performed prior to the “ping” phase of the PathPing test). If this parameter is unassigned then the default value of 3 is used. This parameter applies when the CPE serves as a PathPing client.	-
PathPingTracerouteResolveHostnames	boolean	W	When TRUE, then the IP addresses for the hops discovered by the traceroute are resolved to hostnames using DNS. When FALSE, then only the IP addresses are obtained. The default value for the parameter (i.e. its assumed value when unassigned) is TRUE. This parameter applies when the CPE serves as a PathPing client.	-
PathPingTracerouteMaxHops	unsgined	W	When this parameter is set, it indicates the maximum number of hops to use for the traceroute phase of the PathPing test. When unassigned then the default value of 30 is assumed. This parameter applies when the CPE serves as a PathPing client.	-
PathPingTimeout	unsignedInt (milliseconds)	W	Timeout in milliseconds used for both the traceroute and ping phases of a PathPing test. It represents the amount of time a PathPing client (i.e. performing either traceroute or ping) waits for a reply before determining that the packet is lost. This parameter applies when the CPE serves as a PathPing client.	-
PathPingPort	unsignedInt	W	The port used for the “ping” phase of the PathPing test to intermediate hops when either UDP or TCP transport is selected for PathPingIntermediateHopTransport or to the destination hop when either UDP or TCP transport is selected PathPingDestinationHopTransport. PathPingPort does not apply when ICMP is selected. PathPingPort also represents the port that the device will reply to when functioning as a PathPing server (i.e. when PathPingClientMode = FALSE).	
PathPingIntermediateHopTransport	unsigned[0:3]	W	The transport protocol used to perform the pings to the intermediate hops (i.e. all hops in the path other than PathPingHostTarget) during the ping test phase of the PathPing test. Note that an interation of ping tests is performed to all the hops in the path for each packet size specified by the packet size range (i.e. PathPingMinPacketSize, PathPingMaxPacketSize, and PathPingPacketSizeStepIncrement). PathPingIntermediateHopTransport must be set to an unsigned integer in the range from 0 to 2. When set to 0 then ICMP is used. When set to 1 the UDP is used. When set to 2 then TCP is used. Currently any value greater than 2 is interpreted as setting PathPingIntermediateHopTransport to 0 (i.e. ICMP ping). The default value for this parameter when unassigned is 0. Note that the transport protocol indicated by the parameter only applies to the ping tests performed on each hop prior to the destination hop (PathPingHostTarget) hence the name PathPingIntermediateHopTransport. Note that when this parameter is set to 1 or 2 to indicate the UDP or TCP protocol, respectively, then the pings for each hop are	-

			performed in a manner similar to traceroute. That is, with TTL incrementing from 1 to N-1 for the N-1 intermediate hops along the path from this device to PathPingHostTarget. So when PathPingIntermediateHopTransport = 1 or 2, the device should expect ICMP “ttl timeout exceeded” messages to be returned from the intermediate hops in response to the TTL limited UDP or TCP packets sent from this deivce. This parameter applies when the CPE serves as a PathPing client.	
PathPingDestinationHopTransport	unsigned[0:3]	W	The transport protocol used to perform the pings to the destination hop (i.e. PathPingHostTarget) for each interation of the ping tests phase of the PathPing test. Note that an interation of ping tests is performed to all the hops in the path for each packet size in the range specified by the packet size range (i.e. PathPingMinPacketSize, PathPingMaxPacketSize, and PathPingPacketSizeStepIncrement). PathPingDestinationHopTransport must be set to an unsigned integer in the range from 0 to 3. When set to 0 then ICMP ping is used. When set to 1 the UDP echo is used. When set to 2 then TCP echo is used. When set to 3 the UDPEchoPlus is used. Currently any value greater than 3 is interpreted as setting PathPingDestinationHopTransport to 0 (i.e. ICMP ping). The default value for this parameter when unassigned is 0. Unlike PathPingIntermediateHostTranport, TTL limiting (to trigger TTL timeouts) is not used when PathPingDestinationHop transport is set to 1, 2 or 3, That is, when PathPingDestinationHop is set to 1, 2 or 3, to specify UDP, TCP, or UDPEchoPlus, the destination host (PathPingHostTarget) is assumed to be a complaint server for the specified transport (e.g. UDP echo server, TCP echo server, or UDPEchoPlus server). Note that the UDP, TCP, or UDPEchoPlus packets are sent over port PathPingPort. This parameter applies when the CPE serves as a PathPing client.	-
PathPingNumberOfGroupRepetitions	unsigned	W	The number of packet bursts (i.e. groups of packets having inter-packet spacing of PathPingIntraGroupInterarrivalTime) to send for each hop at each packet size. The ping phase of a PathPing test is comprised of PathPingNumberOfGroupRepetitions * PathPingNumberOfPacketsPerGroup sent to each hop for each packet size in the packet size range. This parameter applies when the CPE serves as a PathPing client.	
PathPingNumberOfPacketsPerGroup	unsigned	W	The number of packets sent as a burst with an inter-packet spacing of PathPingIntraGroupInterarrivalTime. The traffic pattern used during the ping phase of the PathPing test is to send a burst (i.e. group) of packets having an inter-packet spacing of PathPingIntraGroupInterarrivalTime with a spacing between the last packet of burst N and the first packet of burst N+1 (i.e. inter-burst spacing) of PathPingInterGroupInterarrivalTime. This parameter applies when the CPE serves as a PathPing client.	-
PathPingIntraGroupInterarrival	unsigned (millseconds)	W	The interarrival time between two consecutive packets during a burst period of the ping phase of the PathPing test. The traffic pattern used during the ping phase of the PathPing test is to send a burst (i.e. group) of packets having an inter-packet spacing of PathPingIntraGroupInterarrivalTime with a spacing between the last packet of burst N and the first packet of burst N+1 (i.e. inter-burst spacing) of PathPingInterGroupInterarrivalTime. This parameter applies when the CPE serves as a PathPing client.	-

PathPingInterGroupInterarrivalTime	unsigned (millseconds)	W	The interarrival time between the last packet of burst N and the first packet of burst N+1 of the ping phase of the PathPing test. That is, PathPingInterGroupInterarrivalTime denotes the inter-burst (i.e. inter-group) spacing during the ping phase of the PathPing test. The traffic pattern used during the ping phase of the PathPing test is to send a burst (i.e. group) of packets having an inter-packet spacing of PathPingIntraGroupInterarrivalTime with a spacing between the last packet of burst N and the first packet of burst N+1 (i.e. inter-burst spacing) of PathPingInterGroupInterarrivalTime. This parameter applies when the CPE serves as a PathPing client.	-
PathPingMinPacketSize	unsigned (bytes)	W	The minimum packet size to use for the ping phase of the PathPing test. Note that during the ping phase of the PathPing test, pings are performed to each hop identified during the traceroute phase of the PathPing test for each packet size in the range from PathPingMinPacketSize to PathPingMaxPacketSize in steps of PathPingPacketSizeStepIncrement. All of the hops in the path are pinged using one packet size before the next packet size is used. This parameter applies when the CPE serves as a PathPing client.	-
PathPingMaxPacketSize	unsigned (bytes)	W	The maximum packet size to use for the ping phase of the PathPing test. Note that during the ping phase of the PathPing test, pings are performed to each hop identified during the traceroute phase of the PathPing test for each packet size in the range from PathPingMinPacketSize to PathPingMaxPacketSize in steps of PathPingPacketSizeStepIncrement. All of the hops in the path are pinged using one packet size before the next packet size is used. This parameter applies when the CPE serves as a PathPing client.	-
PathPingPacketSizeStepIncrement	unsigned (bytes)	W	The step size used to increase the packet size from its current value to its next value in the range from PathPingMinPacketSize to PathPingMaxPacketSize. If the current packet size + PathPingPacketSizeStepIncrement is greater than PathPingMaxPacketSize, then that packet size is not used. Note that during the ping phase of the PathPing test, pings are performed to each hop identified during the traceroute phase of the PathPing test for each packet size in the range from PathPingMinPacketSize to PathPingMaxPacketSize in steps of PathPingPacketSizeStepIncrement. All of the hops are pinged to using one packet size before the next packet size is used. This parameter applies when the CPE serves as a PathPing client.	
.PathPingConfig.IndividualPacketResults{i}	object	-	This object provides the results from individual PathPing test probes (i.e. packets) collected during the traceroute test and subsequent ping tests associated with a PathPing test.	-
PacketSucess	boolean	-	Indicates that the i-th PathPing packet sent (whether traceroute or ping) was received by the client. When this value is TRUE, then all the remaining PathPingConfig.IndividualPacketResults{i} parameters are valid. Otherwise only the values originally set by the CPE client (e.g. PacketSendTime, TestGenSN, etc …) MAY be set to valid values.	-
PacketSendTime	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the client sent the i-th PathPing packet (whether traceroute or ping) of a PathPing test.	-
PacketRcvTime	dateTime	-	Time in UTC, which MUST be specified to microsecond precision.	-

			For example: 2008-04-09T15:01:05.123456, The dateTime that the client received the i-th PathPing packet (whether traceroute or ping) of a PathPing test, if that packet was received by the CPE client. That is, it represents the return time for the i-th PathPing packet, assuming it was not lost, and is only valid if PacketSucess is TRUE.	
TestGenSN	unsignedInt	-	The TestGenSN value set in the i-th PathPing packet sent when UDPEchoPlus is used as the transport protocol by the CPE client for the ping component to the destination host (i.e. PathPingDestinationHopTransport = 3).	-
TestRespSN	unsignedInt	-	The TestRespSN value returned from the PathPing destination server (i.e. PathPingHostTarget) for the i-th PathPing packet sent when UDPEchoPlus is used as the transport protocol by the CPE client for the ping component to the destination host (i.e. PathPingDestinationHopTransport = 3).. It is only valid if PacketSucess is TRUE.	-
TestRespRecvTimeStamp	unsignedInt	-	The 32bit unsigned integer timestamp (in microseconds) that was set by the UDP Echo Plus server (i.e. PathPingHostTarget) when UDPEchoPlus is used as the transport protocol by the CPE client for the ping component to the destination host (i.e. PathPingDestinationHopTransport = 3) to record the reception time of the i-th UDP Echo Plus packet sent from the CPE client. It is only valid if PacketSucess is TRUE.	-
TestRespReplyTimeStamp	unsignedInt	-	The 32bit unsigned integer timestamp (in microseconds) that was set by the UDP Echo Plus server (i.e. PathPingHostTarget) when UDPEchoPlus is used as the transport protocol by the CPE client for the ping component to the destination host (i.e. PathPingDestinationHopTransport = 3) to record the server reply time of the i-th UDP Echo Plus packet sent from the CPE client. That is, the time that the server returned the UDP Echo Plus packet. It is only valid if PacketSucess is TRUE.	-
TestRespReplyFailureCount	unsignedInt	-	The count value that was set by the UDP Echo Plus server (i.e. PathPingHostTarget) when UDPEchoPlus is used as the transport protocol by the CPE client for the ping component to the destination host (i.e. PathPingDestinationHopTransport = 3) to record the number of locally dropped echo response packets by the server. This count is incremented if a valid echo request packet is received at a UDP EchoPlus server but for some reason cannot be responded to (e.g. due to local buffer overflow, CPU utilization, etc...). It is only valid if PacketSucess is TRUE.	-
Type	string	-	Type is a string that represents the phase of the PathPing test this packet belongs to.That is, the phase the PathPing test was in when the oacket was sent by the CPE. It is set to either “Traceroute” or “Ping” by the CPE. This parameter is set whether PacketSucess is TRUE or FALSE. This parameter applies when the CPE serves as a PathPing client.	-
Transport	string	-	Must be a string from the set of four possible values (“ICMP”, “UDP”, “TCP”, “UDPEchoPlus”) denoting the transport protocol used for this packet. The parameter is set whether the packet was generated during the traceroute phase or ping phase of the PathPing test. Note that this parameter may have different values for different packets throughout the test as a result of the settings for PathPingTracerouteTransport, PathPingIntermediateHopTransport, and PathPingDestinationHopTransport.	

PacketSize	unsignedInt	-	The size in bytes of this packet. This parameter is set whether PacketSucess is TRUE or FALSE. This parameter is set whether the packet was sent during the traceroute phase or ping phase of the PathPing test. This parameter applies when the CPE serves as a PathPing client.	-
Hostname	string(256)	-	The Hostname (or IP address if PathPingTracerouteResolveHostnames = FALSE) that this packet was directed to. It represents the Hostname of either the destination (i.e. PathPingHostTarget) or an Intermediate Host and is set whether this packet was sent during the traceroute or ping phase of the PathPing test. This parameter applies when the CPE serves as a PathPing client.	-

Table 9 (PathPing Test Parameters)

When PathPingEnable = TRUE and PathPingClientMode = TRUE, the device functions as a PathPing client. When PathPingEnable = TRUE and PathPingClientMode = FALSE, the device functions as a PathPing server. When functioning as a PathPing server, the device reflects both Traceroute packets received and any echo test (i.e. ICMP pings, UDPEcho, TCPEcho, or UDPEchoPlus) packets received per the format specified by PathPingTransport. That is, when PathPingTransport = 0, the device reflects ICMP Echo packets (i.e. ICMP pings), when PathPingTransport = 1, the device reflects UDP Echo packets (received on udp port PathPingPort) per RFC 862, when PathPingTransport = 2, the device reflects TCP SYN packets (received on tcp port PathPingPort) with TCP SYN ACKs, and when PathPingTransport = 3, the device functions as a UDPEchoPlus server (for packets received on udp port PathPingPort). When functioning as a UDPEchoPlus server the device performs the necessary timestamping and sequence number insertion functions whenever the packet size is greater than or equal to the minimum required size for UDPEchoPlus (see Figure 3 (UDP Echo Plus Packet Format)).

When the device functions as a PathPing client (i.e. PathPingEnable = TRUE and PathPingClientMode = TRUE), the device first performs a traceroute to the destination host, PathPingHostTarget, and retains a list of all of the hosts discovered along the path of the traceroute. Then it immediately performs pings to each device discovered along the traceroute path for each packet size in the range of PathPingMinPacketSize to PathPingMaxPacketSize in steps of PathPingPacketSizeStepIncrement. For example, assume we discover 10 devices in the path from the BHR to Host B (where Host B = PathPingHostTarget). Then if PathPingMinPacketSize = 40, PathPingMaxPacketSize = 1000, and PathPingPacketSizeStepIncrement = 100, we would perform pings to each of the 10 hosts in the path using a ping packet size of 40 bytes (i.e. iteration 1), followed by pings to each of the 10 hosts in the path using a ping packet size 140 bytes (i.e. iteration 2), followed by pings to each host in the path using a ping packet size of 240 bytes (i.e. iteration 3), and so on up to a packet size of 940 bytes. However, the pings would not be performed for the packet size of 1040 bytes, since that value is greater the PathPingMaxPacketSize of 1000 bytes. Note if PathPingMinPacketSize and PathPingMaxPacketSize are set to the same value than a single packet size is used for the pings. That is, when PathPingMinPacketSize = PathPingMaxPacketSize a single iteration of pings is performed with packet size = PathPingMinPacketSize = PathPingMaxPacketSize to each device discovered in the traceroute. Similarly, if

PathPingPacketSizeStepIncrement is equal to zero or is unassigned then a single iteration of pings is performed to each hop with packet size = PathPingMinPacketSize.

Note that the total number a packets generated "to each hop in the path" per packet size iteration is PacketsGeneratedPerIteration = PathPingNumberOfGroupRepetitions * PathPingNumberOfPacketsPerGroup and the total number of packet size iterations is PacketSizeIterations = IntegerPart[(PathPingMaxPacketSize – PathPingMinPacketSize)/ PathPingPacketSizeStepIncrement]. Thus the total number of packets generated during the ping phase of a PathPing test NumberOfHops*PacketsGeneratedPerIteration * PacketSizeIterations. However, as previously mentioned, when PathPingMaxPacketSize = PathPingMinPacketSize or when PathPingPacketSizeStepIncrement = 0 (or PathPingPacketSizeStepIncrement is invalid or unassigned), then a single iteration at packet size = PathPingMinPacketSize is performed.

For each packet size, the pings performed to each host detected from the traceroute (i.e. each hop) are performed using an interarrival pattern dictated by the parameters PathPingNumberOfGroupRepetitions, PathPingNumberOfPacketsPerGroup, PathPingIntraGroupInterarrivalTime and PathPingInterGroupInterarrivalTime. The traffic generation pattern is depicted in the following Figure:

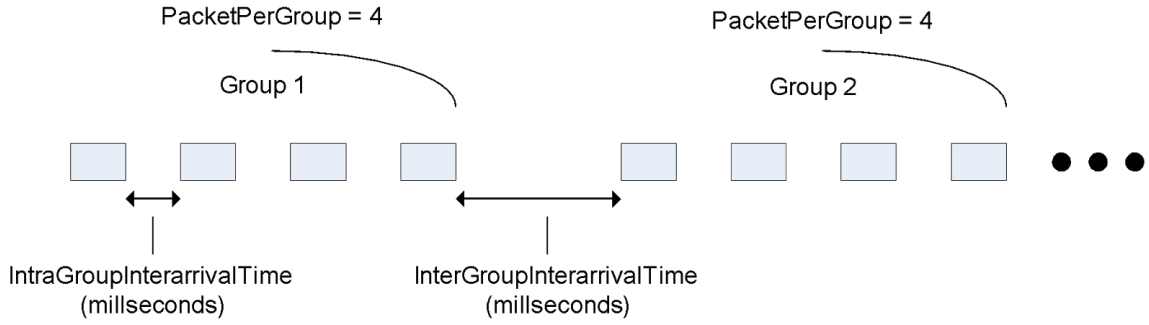

Figure 27 (PathPing Traffic Generation Depiction)

The traffic is generated in terms of groups of closely spaced packets which have a larger spacing between the groups. The spacing (i.e. interarrival times) between packets within the same group is called the IntraGroupInterarrivalTime and the spacing between the last packet of group N and the first packet of group N+1 is called the InterGroupInterarrivalTime. The number of packets contained within a group is designated by the PathPingNumberOfPacketsPerGroup parameter. This parameter set allows one to send the packets as packet pairs, or in groups of 3 packets, or groups or 4 packets, etc … If the PathPingNumberOfPacketsPerGroup is set to 1, then the IntraGroupInterarrival parameter is ignored and one obtains a simple constant bit rate packet train (i.e. a stream of packets having constant interarrival time = InterGroupInterarrivalTime).

Note that when PathPingTracerouteTransport is set to 1 (i.e. UDP), or 2 (i.e. TCP) the traceroutes to the intermediate devices along the path are all performed in a manner based

on the TTL settings. That is, the TTL value in the UDP or TCP packet is set such that TTL will expire at each device along the path. Though a UDP or TCP packet is sent, an ICMP "time exceeded" response packet should be received. The objective for the BHR is to just measure round trip times (of packet send times to the time the responses to those packets are received) and the packet loss events (i.e. for non returned packets). Note that the same approach is used for the intermediate hops during the ping phase of the PathPing test when PathPingIntermediateTransport is set to 1 (i.e. UDP), or 2 (i.e. TCP). That is, TTL values are limited to the (intermediate) hop number to ensure ICMP "time exceeded" response packets are received at the hops being pinged. The difference between the traceroute and ping phases for the intermediate hops, however, is the traffic patterns used for the traceroute traffic generation and ping traffic generation from the BHR. However, when PathPingIntermediateTransport = 0 (i.e. ICMP), the actual ICMP requests are generated with no limit set in the TTL field. That is, standard pings are used to when PathPingIntermediateTransport = 0. Also, no TTL limiting is used for pinging the destination hop (PathPingHostTarget) as the destination hop is assumed to be a compliant UDP Echo, or TCP Echo, or UDPEchoPlus server if such transport selected for PathPingDestinationTransport. Also, as is the case for intermediate hops, if ICMP is selected for PathPingDestinationTransport (i.e. a value of 0), then standard ICMP pings are performed to the destination hop during the ping phase of the PathPing test.

[G.2] When the BHR is functioning as a PathPing client (i.e. when PathPingEnable = TRUE and PathPingClientMode = TRUE). A TestResult message is sent to the test server via a TCP connection after a PathPing test completes, just as is the case for throughput test results. Because the PathPing test is not a throughput test, it requires some modifications to the TestResult message field definitions. Let us first define a "PathPing data record" as a 320 byte data structure comprised of the following fields:

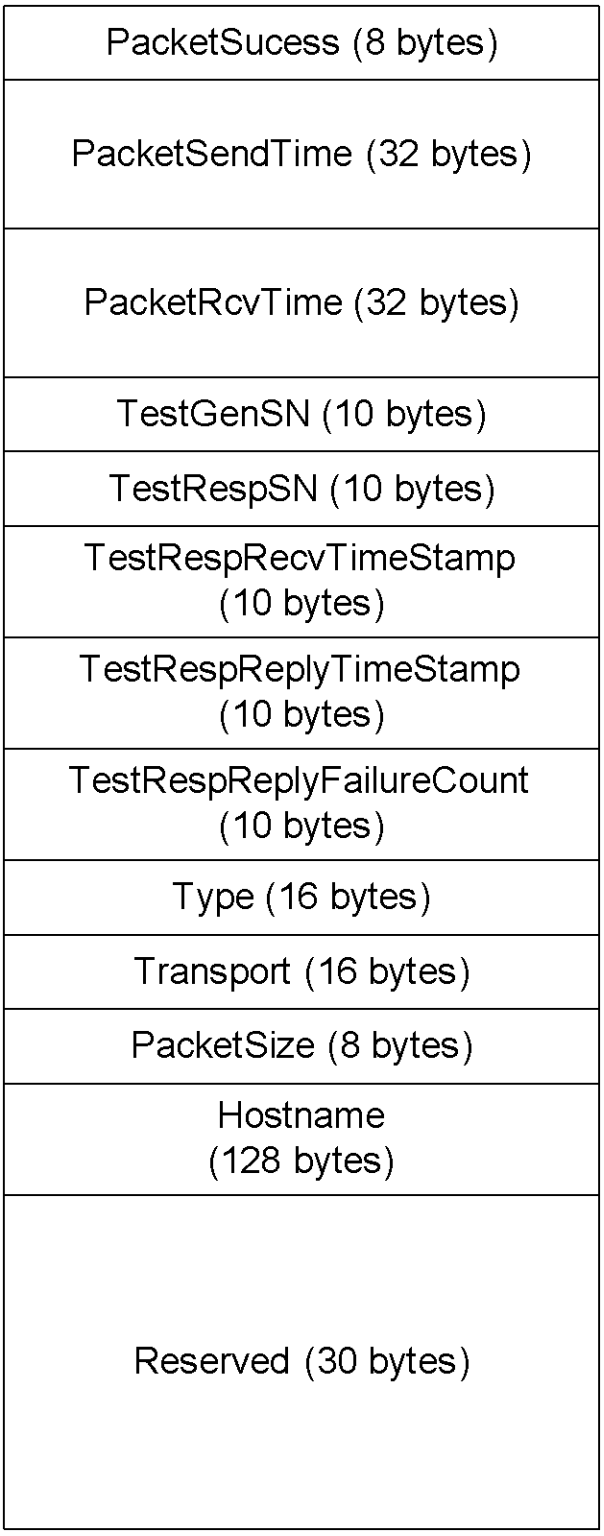

Figure 28 (PathPing Data Record)

The 8 byte PacketSucess field is a string value set to either “TRUE” or “FALSE” according to whether or not a PathPing packet (i.e. either a traceroute and ping packet in the PathPing test) is successfully sent to and returned from a host in the path. The 32 byte PacketSendTime and PacketRcvTime fields (as defined in Table 9 (PathPing Test Parameters)) are the dateTime string values in UTC specified to microsecond precision (e.g. 2008-04-09T15:01:05.123456). These timestamps represent the times when the PathPing packet was sent to and received from PathPingHostTarget or an intermediate host, respectively. Note that if the packet was a UDPEchoPlus packet sent to and replied to a UDPEchoPlus server at PathPingHostTarget (i.e. PathPingHostTarget is a UDPEchoPlus server), then the remaining fieldsTestGenSN, TestRespSN, TestRespRecvTimestamp, TestRespReplyTimestamp, and TestRespReplyFailure fields as defined in Table 9 (PathPing Test Parameters) are also populated (assuming PacketSucess is TRUE). These fields are populated using a format identical to the field formats used for the UDPEchoPlus data record (see Figure 21 (UDPEchoPlus Data Record)). Note that a PathPing data record corresponds to a single PathPing packet. A PathPing data record is constructed and appended to the PathPing Test Result message for every PathPing packet sent during the test (whether traceroute or ping).

A PathPing test is comprised of a traceroute test immediately followed by a sequence of ping tests. During the ping test phase, an iteration of ping tests for each packet size in the packet size range is performed to each hop identified by the traceroute test. A PathPing data record is supplied for every traceroute packet and every ping packet generated during the test. The “Type” field is populated with a string value either equal to “Traceroute” or “Ping”. This field allows the test server to discern during what phase of the test the packet corresponding to the data record was generated. Depending on the transport protocol settings assigned to PathPingIntermediateHopTransport and/or PathPingDestinationHopTransport, the transport protocol used for the pings to the intermediate and destination hops, respectively, may be ICMP (i.e. standard ping), or UDP, or TCP, or UDPEchoPlus (which only applies to the destination hop). For all of the transports used, however, the Type field will use the string “Ping” to denote the packet was sent during the ping/echo phase of the PathPing test (regardless of the transport used) or “Traceroute” to indicate the packet was sent during the traceroute phase of the test. The Type field is populated whether the PacketSucess field is TRUE or FALSE.

The Transport field can be populated with one of four possible string values to indicate the transport used for the packet corresponding to this data record: “ICMP”, “UDP”, “TCP”, or “UDPEchoPlus”. One of these four strings must be used to populate the Transport field whether the packet was sent during the traceroute phase or ping phase of the PathPing test (depending on which transport was used for the packet). The Transport field is populated whether the PacketSucess field is TRUE or FALSE.

The PacketSize field is populated with a “string” value (though the parameter is an unsigned integer in Table 9 (PathPing Test Parameters) indicating the size of the

traceroute or ping packet corresponding to this data record and the Hostname field is populated with a string denoting the hostname or IP address that the traceroute or ping packet corresponding to this data record was sent to. That is, the Hostname field should always correspond to the particular host in the path, intermediate or destination, which the packet corresponding to this data record was sent to.

Because the UDPEchoPlus specific data fields in the PathPing data record only apply to the destination host (i.e. PathPingHostTarget), the fields are all set to zero's for the pings performed to the intermediate hosts or when PathPingDestinationHostTransport is not set to UDPEchoPlus. The UDPEchoPlus fields are also set to zero during the traceroute phase of the test. The UDPEchoPlus specific fields are TestGenSN, TestRespSN, TestRespRecvTimestamp, TestRespReplyTimestamp, and TestRespReplyFailureCount. So if a PathPing test were performed between a BHR and Host B with one intermediate host, say Host A, then the BHR would first perform a traceroute to PathPingHostTarget = Host B and obtain the list of hosts: Host A and Host B. If PathPingDestinationTransport = UDPEchoPlus, the BHR would first perform pings to Host A using the protocol specified by PathPingIntermediateHostTransport. For example, if PathPingIntermediateHostTransport = 1 (i.e. UDP), then UDP packets with a TTL value = 1 would be used for "pinging" Host A. Altenatively, if PathPingIntermediateHostTransport = 0, then standard ICMP Echo pings would be used to "ping" Host A. In either case, all of the pings to Host A, or any intermediate device would result in a PathPing data record having all of the UDPEchoPlus specific parameters to zero. However, when pinging the destination device (Host B in this case), the UDPEchoPlus option is available. So if PathPingDestinationTransport = 3 (i.e. UDPEchoPlus) then when "pinging" the destination device (i.e. PathPingHostTarget), UDPEchoPlus complaint echos (serving as "pings") are performed to Host B (the target host in this example) and UDPEchoPlus fields in the resulting PathPing data records are set per the test results for each UDPEchoPlus packet. That is, a UDPEchoPlus test using the traffic pattern defined by PathPingNumberOfGroupRepetitions, PathPingNumberOfPacketsPerGroup, PathPingIntraGroupIntervarrivalTime , PathPingInterGroupInterarrivalTime per Figure 27 (PathPing Traffic Generation Depiction) is the final set of pings performed between the BHR and the PathPingHostTarget for each packet size in the range between PathPingMinPacketSize and PathPingMaxPacketSize (in steps of PathPingSizeStepIncrement).

The PathPing data records are appended to the PathPing Test Result message and there is a PathPing data record for every packet sent from the BHR during the PathPing test including both the traceroute packets and ping packets. The PathPing Test Result message is defined as follows:

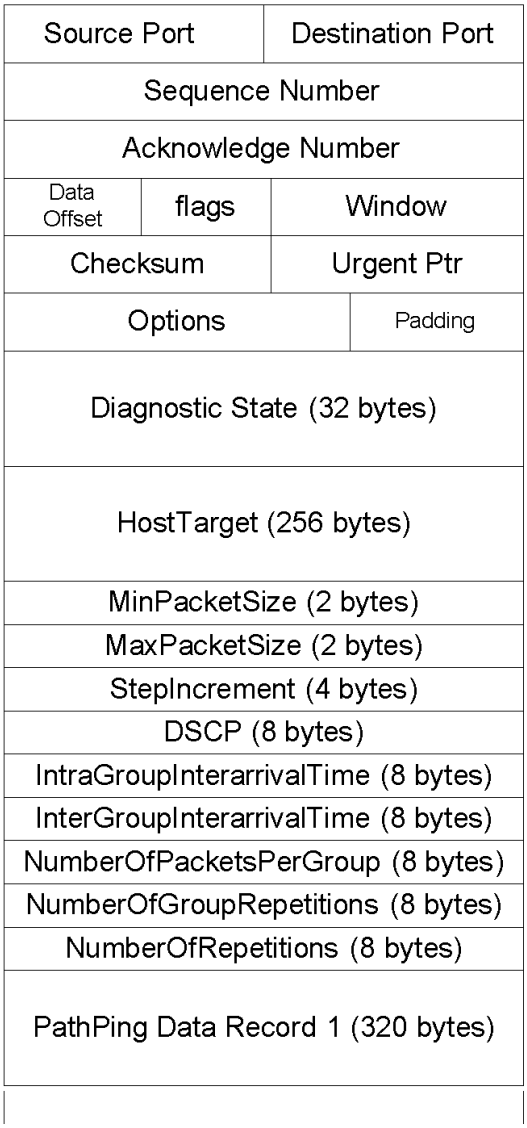

Figure 29 (PathPing Test Result Message)

The DiagnosticState field is populated with a string value per the DiagnosticState parameter defined in Table 9 (PathPing Test Parameters). The HostTarget field is populated with the same string value as PathPingHostTarget from Table 9 (PathPing Test Parameters). The MinPacketSize, MaxPacketSize, StepIncrement and DSCP fields are all unsigned integers populated with the values of MinPacketSize, MaxPacketSize, StepIncrement, and DSCP, respectively, as defined in Table 9 (PathPing Test Parameters). The InterGroupInterarrivalTime, IntraGroupInterarrivalTime, NumberOfPacketsPerGroup, and NumberOfGroupRepetitions fields are populated per the PathPingInterGroupInterarrivalTime, PathPingIntraGroupIntraArrivalTime, PathPing NumberOfPacketsPerGroup, and PathPing NumberOfGroupRepetitions parameters respectively as defined in Table 9 (PathPing Test Parameters). However, the

InterGroupInterarrivalTime, IntraGroupInterarrivalTime, NumberOfPacketsPerGroup, and NumberOfGroupRepetitions fields are populated as string value representations of these parameters (though they are actually unsigned integer types in the table). The final field in the header, NumberOfRepetitions, is populated with the total number of packets generated (traceroute + ping packets) for the PathPing tests and represents the total number of data records that will be appended to the PathPing test result message. Assuming the number of hops identified from the traceroute phase of the PathPing test is NumHops, then NumberOfRepetitions should equal NumHops * (PathPingTracerouteQueriesPerHop + M), where M = the number of ping packets per hop. Note M is computed by M = PathPingNumberOfGroupRepetitions * PathPingNumberOfPacketsPerGroup * S), where S is the number of packet sizes used during the ping phase of the PathPing test. S is computed by S = PacketSizeIterations = IntegerPart[(PathPingMaxPacketSize – PathPingMinPacketSize)/ PathPingPacketSizeStepIncrement].

[G.3] **Theory of Operation**: The client mode PathPing test operates by the BHR first performing a traceroute to PathPingHostTarget using the PathPingTracerouteTransport setting as the transport protocol. The number of packets sent to each hop is dictated by PathPingTracerouteNumberOfQueries and the packet size of each traceroute packet is dictated by PathPingTraceroutePacketSize. Note that the traceroute phase of the PathPing test uses a single packet size. Also though the traffic pattern for the traceroute phase is not controlled by any PathPing parameters (i.e. there is no interarrival time specified for the traceroute component of the PathPing test), a typical traceroute implementation operates as follows. After sending a traceroute request packet (using the transport specified in PathPingTracerouteTransport), wait for PathPingTimeout milliseconds for a reply and once the reply is received or PathPingTimeout expires, then immediately send the next traceroute request packet.

If PathPingTracerouteOnly = TRUE, then terminate the test after the traceroute phase is complete. Otherwise, if PathPingTracerouteOnly = FALSE, then the traceroute phase of the PathPing test is followed by the ping phase of thtest. During the ping phase of the PathPing test, a sequence of PacketsGeneratedPerIteration * PacketSizeIterations pings is sent to each hop in the path (i.e. the path inferred from the traceroute phase of the PathPing test) and the traffic pattern used for the ping phase of the PathPing test to each hop is specified by PathPingNumberOfGroupRepetitions, PathPingNumberOfPacketsPerGroup, PathPingIntraGroupInterarrivalTime and PathPingInterGroupInterarrivalTime as depicted in Figure 27 (PathPing Traffic Generation Depiction). Starting with a packet size equal to PathPingMinPacketSize, pings are performed to each hop in the path using the protocol specified by PathPingIntermediateHopTransport for each hop in the path except for the final (i.e. destination) hop. Note TTL limiting (as done in traceroute) is used for each PathPingIntermediateHopTransport protocol setting other than ICMP (where standard ICMP echo pings are used), though the traffic pattern is different for the ping phase of the PathPing test. For the destination hop (i.e. PathPingHostTarget), the

PathPingDestinationHopTransport is used for the pings to PathPingHostTarget and TTL limiting is not used for the non-ICMP protocols since the destination hop is assumed to be a complaiant server to the PathPingDestinationHopTransport protocol selected. After the pings are performed to the destination hop, the next packet size is computed according to “next packet size” = “current packet size” + PathPingPacketSizeStepIncrement, and if “next packet size” is less than or equal to PathPingMaxPacketSize then the ping tests are performed again using the new packet size for the intermediate hops followed by the pings to the destination hop using PathPingIntermediateHopTransport and PathPingDestinationHopTransport, respectively. Each of the pings, for each packet size of the ping phase of the PathPing test, use the traffic pattern specified by PathPingNumberOfGroupRepetitions, PathPingNumberOfPacketsPerGroup, PathPingIntraGroupInterarrivalTime and PathPingInterGroupInterarrivalTime as depicted in Figure 27 (PathPing Traffic Generation Depiction).

Appendix H – Passive Usage Monitoring

This section provides a specification for a Passive Usage Monitoring test. The Passage Usage Monitoring test provides a high resolution capture of the aggregate and/or filtered usage on a selected port of the BHR for a preselected duration of time. Counts of the aggregate and/or filtered usage are performed in terms of bytes and packets in time increments down to 1 sec. The high resolution counts are captured and stored at the BHR and forwarded to the test server upon completion of the test.

[H.1] We propose adding the following Passive Usage Monitoring test parameters to the device at the device path location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.PassiveUsageMonitoring for the purpose of incorporating the test into the TR-143 test suite – DiagnosticState (string), Interface (string[256]), TestDuration (unsignedInt), SamplingInterval (unsignedInt). In addition we add InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.PassiveUsageMonitoring.Filter{i} which add qualifiers to the collection of byte and packet counts (where i is in the range from 0 <= i <= 3). That is, the BHR can simultaneously measure four sets of byte and packet counts, one set for each of the four indexed filter parameter sets. Each filter parameter set consists of the following parameters: ProtocolType, UpstreamPort, DownstreamPort, DSCP, and EthernetPriority. A set of test results is included for each filter at the location: InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.PassiveUsageMonitoring.Filter{i}.Results. which contains the test result parameters – StartTime, EndTime, NumberOfDataSamples and an individual set of byte and packet counts is stored for each result at InternetGatewayDevice.X_ACTIONTEC PerformanceDiagnostics.PassiveUsageMonitoring.Filter{i}.Results.IndividualResults{j}

which contains UpstreamBytes (unsignedInt), UpstreamPackets(unignedInt), DownstreamBytes (unsignedInt), and DownstreamPackets (unsignedInt). The Passive Usage Monitoring parameters are illustrated in the following Table:

Name	Type	Write	Description	Object Default
.PassiveUsageMonitoring.	object	-	This object allows the CPE to be configured to perform the Passive Usage Monitoring (PUM) test. The PUM test functions by the CPE measuring downstream and upstream byte and packet counts for all packets that cross the Interface parameter for four simultaneous filterting criteria. That is, the counts for four bi-directional flows are measured simultaneously in terms of both bytes and packets. The four filtering criteria are determined by the parameter settings in the .Filter{i} indexed sub-obects.	-
DiagnosticState	string	W	Indicates availability of diagnostic data when the device functions as a PassiveUsageMonitoring (PUM) agent. The value may be one of: “None” “Requested” “Complete” “Error_Internal” “Error_Other” If the ACS sets the value of this parameter to Requested, the CPE MUST initiate a single PassiveUsageMonitoring test. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic. When the each test is completed, the value of this parameter MUST be either Complete (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Complete, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result	-

			in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	IP-layer interface over which the CPE MUST perform the Passive Usage Monitoring test. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE perform this test on the default Interface which is the Broadband Interface (i.e. WAN port) of the device.	-
TestDuration	unsginedInt (seconds)	W	The duration of time in seconds to perform the Passive Usage Montoring test. Note a Passive Usage Monitoring test having TestDuration = 3600 (i.e. 1 hour) and SamplingInterval = 1 will have 3600 individual data samples (i.e.where each individual data sample contains byte and packet counts for both the upstream and downstream directions) for each filter enabled.	-
SamplingInterval	unsignedInt (seconds)	W	The granular period of time upon which each data sample is updated (i.e.where each individual data sample contains a count for UpstreamBytes, UpstreamPackets, DownstreamBytes, DownstreamPackets). The parameter can be set to a minimum value of 1 (second). The number of data samples collected for a Passive Usage Monitoring test when all four filters are enabled is: Ceiling [TestDuration/SamplingInterval]*4, where the 4 indicates the maximum number of filters that can simultaneously be enabled. Each filter has its own set of data sample results, where each data sample result is comprised of 4 parameters (UpstreamBytes, UpstreamPackets, DownstreamBytes and DownstreamPackets). Note that if TestDuration is set less than SamplingInterval, or SamplingInterval is zero or unassigned, than the test will terminate after TestDuration expires and the results will correspond to a single sampling interval equal to TestDuration.	
.PassiveUsageMonitoring .Filters{i}	object	-	This object includes the parameters used for defining up to four (i.e. 0 <= i <= 3) filter sets for the collection of Passive Usage Monitoring results.Since static values for packet fields matching are used for defining the filters, the device’s QoS hardware may be leveraged to implement the filtering. When enabled, the upstream and downstream, bytes and packets, for each enabled filter must be measured concurrently.	-
Enable	boolean	W	When set to TRUE, the filter conditions are used to determine a match to specified traffic fields for those packets to be included in the counts and data samples are measured for the qualifying packets. Each data sample is comprised of bytes and packets measured for both the Upstream and Downstream directions (DownstreamBytes, DownstreamPackets, UpstreamBytes, and UpstreamPackets) on the interface specified by the Interface parameter for those packets that meet the conditions specified by this filter. That is, when Enable = TRUE, the results for this filter are included in the collected during the Passive Usage Monitoring test. When Enable = FALSE, this filter is ignored and no data samples are collected for this filter (though the other filters may still be	-

			enabled and collect data since each filter operates independently).	
Include DSCP	boolean	W	When TRUE, the the DSCP parameter setting (below) is included in the filter criterea. When set to FALSE, the DSCP setting is ignored and upstream and downstream, bytes and packets, are included in the results counts regardless of there DSCP setting. Thus setting the parameter to FALSE implies a wildcard condition for the DSCP filter.	-
DSCP	unsignedInt [0:63]	W	DiffServ codeng point to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, IP packets that do not have a DSCP codepoint equal to this parameter setting are excluded by from the results of this filter. By default the CPE SHOULD set this value to zero.	-
Include EthernetPriority	boolean	W	When TRUE, the the Ethernet priority parameter setting (below) is included in the filter criterea. When set to FALSE, the EthernetPriority setting is ignored and upstream and downstream, bytes and packets, are included in the results counts regardless of their Ethernet Priority setting. Thus setting the parameter to FALSE implies a wildcard condition for the Ethernet Priority filter.	-
EthernetPriority	unsignedInt [0:63]	W	Ethernet Priority value to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, Ethernet frames that do not have a Ethernet Priority field equal to this parameter setting are excluded from the results of this filter. By default the CPE SHOULD set this value to zero.	-
Include Protocol Type	boolean	W	When TRUE, the the Protocol Type parameter setting (below) is included in the filter criterea. When set to FALSE, the Protocol Type setting is ignored and upstream and downstream, bytes and packets, are included in the results counts regardless of their Protocol Type field setting. Thus setting the parameter to FALSE implies a wildcard condition for the Protocol Type filter.	-
Protocol Type	unignedInt	W	Protocol Type value to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, IP packets that do not have a Protocol Type field equal to this parameter setting are excluded from the results of this filter. By default the CPE SHOULD set this value to zero.	-
Include Source Port Range	boolean	W	When TRUE, the the Min Source Port and Max Source port parameter settings (below) are included in the filter criterea. When set to FALSE, the Source Port range settings are ignored and upstream and downstream, bytes and packets, are included in the results counts regardless of their Source Port field settings in the IP packet. Thus setting the parameter to FALSE implies a wildcard condition for the Source Port range filter.	-
Min Source Port	unignedInt	W	The minimum Source Port value in the range to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, IP packets having UDP or TCP transport that do not have a Source Port field within the range of Min Source Port to Max Source Port are excluded from the results of this filter. By default the CPE SHOULD set this value to zero.	-
Max Source Port	unignedInt	W	The maximum Source Port value in the range to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, IP packets having UDP or TCP transport that do not have a Source Port field within the range of Min Source Port to Max Source Port are excluded from the results of this filter. A single Source Port filter criterea is created by setting Min Source Port equal to Max Source Port. By default the CPE SHOULD set this value to 65535.	-
Include Destination Port Range	boolean	W	When TRUE, the the Min Destination Port and Max Destination port parameter settings (below) are included in the filter criterea. When set to FALSE, the Destination Port	-

			range settings are ignored and upstream and downstream, bytes and packets are included in the results counts regardless of their Destination Port field settings in the IP packet. Thus setting the parameter to FALSE implies a wildcard condition for the Destination Port range filter.	
Min Destination Port	unignedInt	W	The minimum Destination Port value in the range to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, IP packets having UDP or TCP transport that do not have a Destination Port field within the range of Min Destination Port to Max Destination Port are excluded from the results of this filter. By default the CPE SHOULD set this value to zero.	-
Max Destination Port	unignedInt	W	The maximum Destination Port value in the range to be used for this filter (inclusive) by the Passive Usage Montoring test. That is, IP packets having UDP or TCP transport that do not have a Destination Port field within the range of Min Destination Port to Max Destination Port are excluded from the results of this filter. . A single Source Port filter criterea is created by setting Min Destination Port equal to Max Destination Port.By default the CPE SHOULD set this value to 65535.	-
.PassiveUsageMonitoring .Filter{i}.Results	object	-	The test results collected for Filter{i} of the Passive Usage Monitoring test.	-
StartTime	dateTime	-	Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the Passive Usage Monitoring test began (for this filter). It MUST equal the start time of the first SamplingInterval for this filter. Since the data for each, of the up to four, filters are collected simultaneously, the value of this timestamp should be virtually equal to the StartTime for the remaing filters.	-
EndTime	dataTime		Time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456, The dateTime that the Passive Usage Monitoring test ended (for this filter). It MUST equal the end time of the last SamplingInterval for this filter. Since the data for each, of the up to four, filters are collected simultaneously, the value of this timestamp should be virtually equal to the EndTime for the remaing filters.	-
NumberOfDataSamples	unsigned	-	The number of data samples collected between StartTime and EndTime for this filter during the Passive Usage Monitoring test. It denotes the number of indexed values, j, to expect in the .PassiveUsageMonitoring.Filter{i}.Results.IndividualResults{j}. sub-object.That is, it should equal the max value of j minus 1 (since j begins at zero).	
.PassiveUsageMonitoring .Filter{i}.Results.IndividualResults{j}	object	-	This object includes the indexed byte and packet count results for this fitler subobject. Note that the value of j will range from 0 to Ceiling [TestDuration/SamplingInterval]. Ceiling is used in case the SamplingInterval does not divide evenly into TestDuration.	-
DownstreamBytes	unsignedInt	-	The number of bytes measured by this filter in the downstream direction at Interface during the j-th SamplingInterval.	-
DownstreamPackets	unsignedInt	-	The number of packets measured by this filter in the downstream direction at Interface during the j-th SamplingInterval.	-
UpstreamBytes	unsignedInt	-	The number of bytes measured by this filter in the upstream direction at Interface during the j-th SamplingInterval.	-

UpstreamPackets	unsignedInt	-	The number of packets measured by this filter in the upstream direction at Interface during the j-th SamplingInterval.	-

Table 10 (Passive Usage Monitoring Test)

[H.2] After the BHR executes a Passive Usage Monitoring test, a TestResult message is sent to the test server via a TCP connection after the test completes, just as is the case for the throughput test results. Because the Passive Usage Monitoring test is not a throughput test, however, it requires some modifications to the TestResult message field definitions. Let us first define a "PassiveUsageMonitoring data record" as a 40 byte data structure comprised of the following fields:

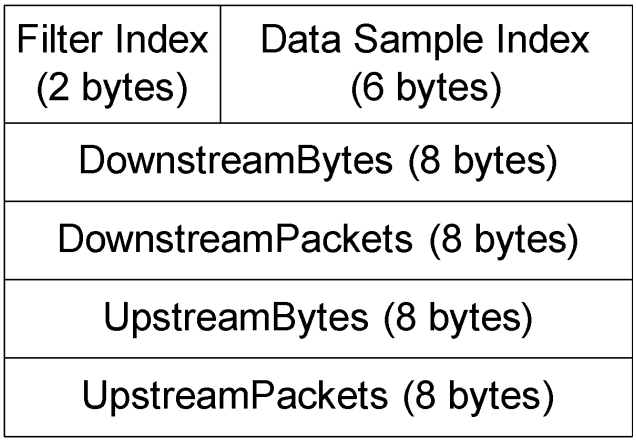

Figure 30 (PassiveUsageMonitoring Data Record)

The 2 byte "Filter Index" field represents the value of the index, i, for the i^{th} filter this data record belonds to (where 0 <= i <= 3 since up to four filters may be enabled for simultaneous data collection). The "Filter Index" field is formatted as an unsigned integer. The 6 byte "Data Sample Index" field represents the value of the index, j, for the j^{th} data sample. It is incremented (starting from zero) by 1 for each data record, and is also formatted as an unsigned integer. For example, a Passive Usage Monitoring test with TestDuration = 10 (seconds) and SamplingInterval = 1 (second) will result in 10 data samples each measured in a 1 second period. Thus, each enabled filter, the index 'j' will range from 0 to 9. The remaining 64 bit (i.e. 8 bytes) fields represent the number of DownstreamBytes, DownstreamPackets, UpstreamBytes, and UpstreamPackets measured, respectively, during the j^{th} sample interval (where j is indicated by "Data Sample Index") on the filter indicated by "Filter Index". Each of the values for DownstreamBytes, DownstreamPackets, UpstreamBytes, and UpstreamPackets are formatted as 64 bit unsigned integers.

Let us now define a "Passive Usage Monitoring Filter Record" as follows:

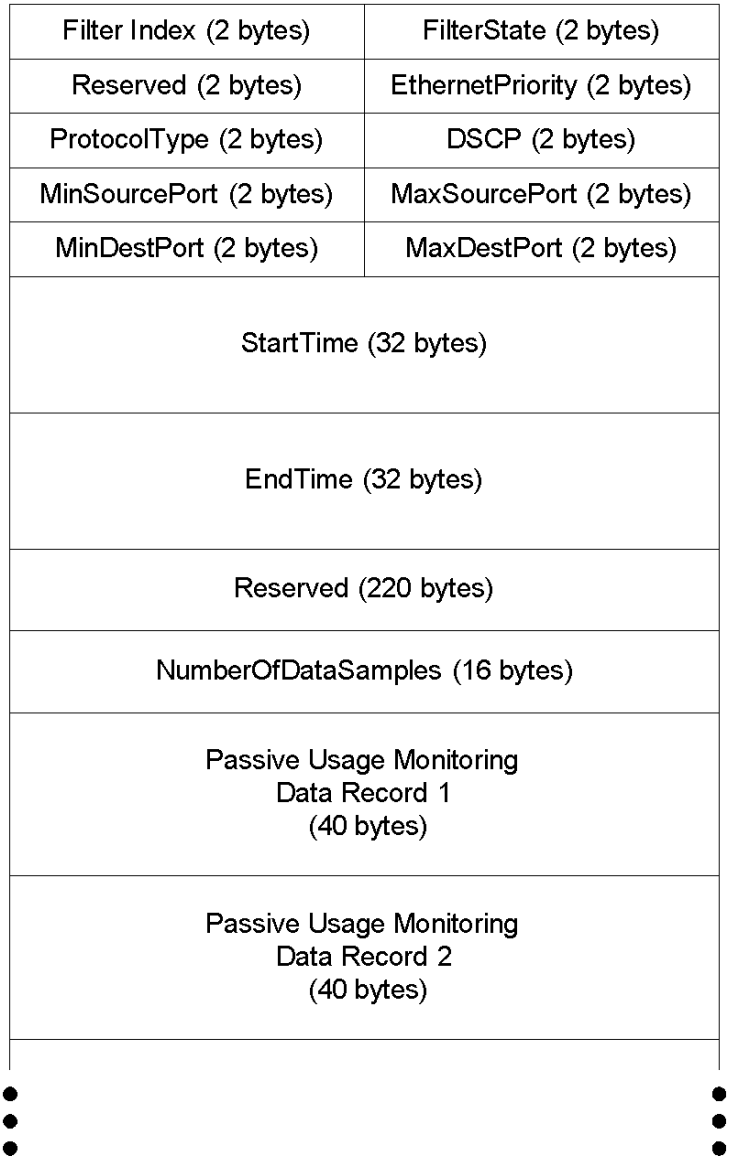

Figure 31 (Passive Usage Monitoring Filter Record – Part1)

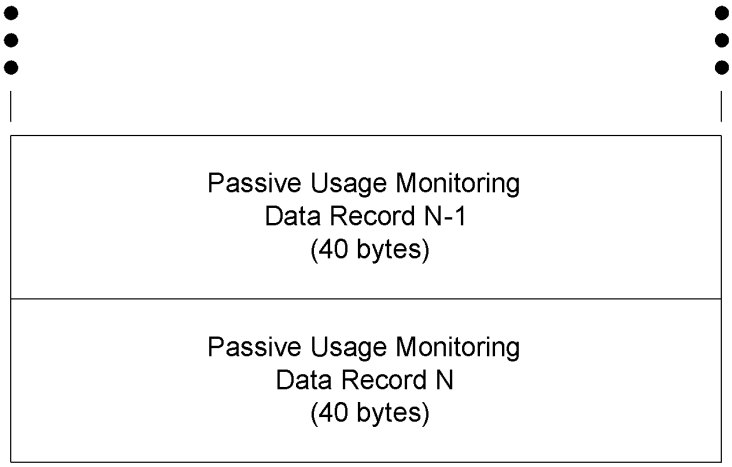

Figure 32 (Passive Usage Monitoring Filter Record - Part 2)

Note that the Passive Usage monitoring Filter Record consists of a 320 byte header with N successive Passive Usage Monitoring Data Records appended, where N = 'NumberOfDataSamples' as illustrated in the last field of the Filter Record header. All of the Data Records appended to a Filter Record correspond to that particular filter, so the 2 byte 'Filter Index' field in the Filter Record header should have the same value as the 'Filter Index' field of each of the appended Data Records. The 2 bytes 'Filter State' field is defined as follows:

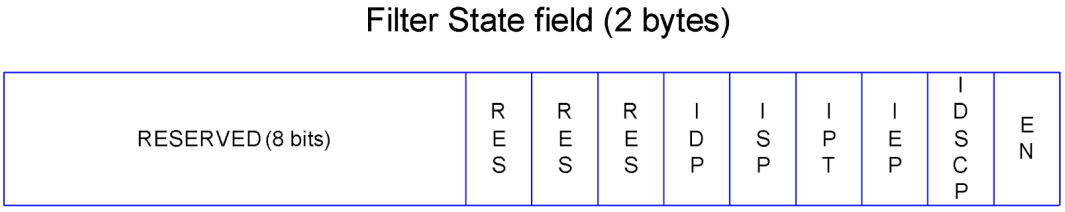

Figure 33 (Filter State field)

The least significant bit is the 'Enable' bit. It is set to 1 when the filter Enable parameter in Table 10 (Passive Usage Monitoring Test) is TRUE and 0 when the Enable parameter is FALSE. Similarly, the IDSCP bit, IEP bit, IPT bit, ISP bit, and IDP bit is set to 1 when, the Include DSCP, Include Ethernet Priority, Include Protocol Type, Include Source Port, Include Destination Port paramters, respectively, is set to TRUE. Otherwise, if the parameter is set to FALSE, the corresponding bit is set to 0. These bits indicate which packet fields were included as part of the traffic match which define the filter. The remaining 11 bits in the 'Filter State' field are reserved for future use.

The next field in the Filter Record is a 2 byte Reserved field, with the following 7 fields each being 2 byte fields that are populated with the parameter values having the same name in Table 10 (Passive Usage Monitoring Test). That is, the 'EthernetPriority', 'DSCP', 'ProtocolType', 'MinSourcePort', 'MaxSourcePort', 'MinDestPort', and 'MaxDestPort' fields are populated with the values for Ethernet Priority, DSCP, Protocol Type, Min Source Port, Max Source port, Min Destination Port, and Max Destination port as depicted in Table 10 (Passive Usage Monitoring Test), and each of these fields is formatted as unsigned integers. Similary, the 32 byte 'StartTime' and 'EndTime' fields are populated as depicted in Table 10 (Passive Usage Monitoring Test), using a dataTime string format with microsecond resolution (e.g. 2008-04-09T15:01:05.123456). The 'EndTime' field is followed by a 220 byte 'Reserved' field, which is in turn followed by a 16 byte 'NumberOfDataSamples' field which indicates the number of Passive Usage Monitoring Data Records which follow. The NumberOfDataSamples parameter is defined as an unsigned integer in the Table, but the 'NumberOfDataSamples' field is formatted as string. The byte length of a Filter Record is indeterminant since it is not know apriori how many Data Records it will contain but the 'NumberOfDataSamples' field provides this quantity of Data Records so that the Filter Record can correctly be parsed at the test server.

Now that the Filter Record and its constituent Data Recrods have been defined, we can now deinfe the TestResult message for the Passive Usage Monitoring test. The TestResult message for the Passive Usage Monitoring test has the following format:

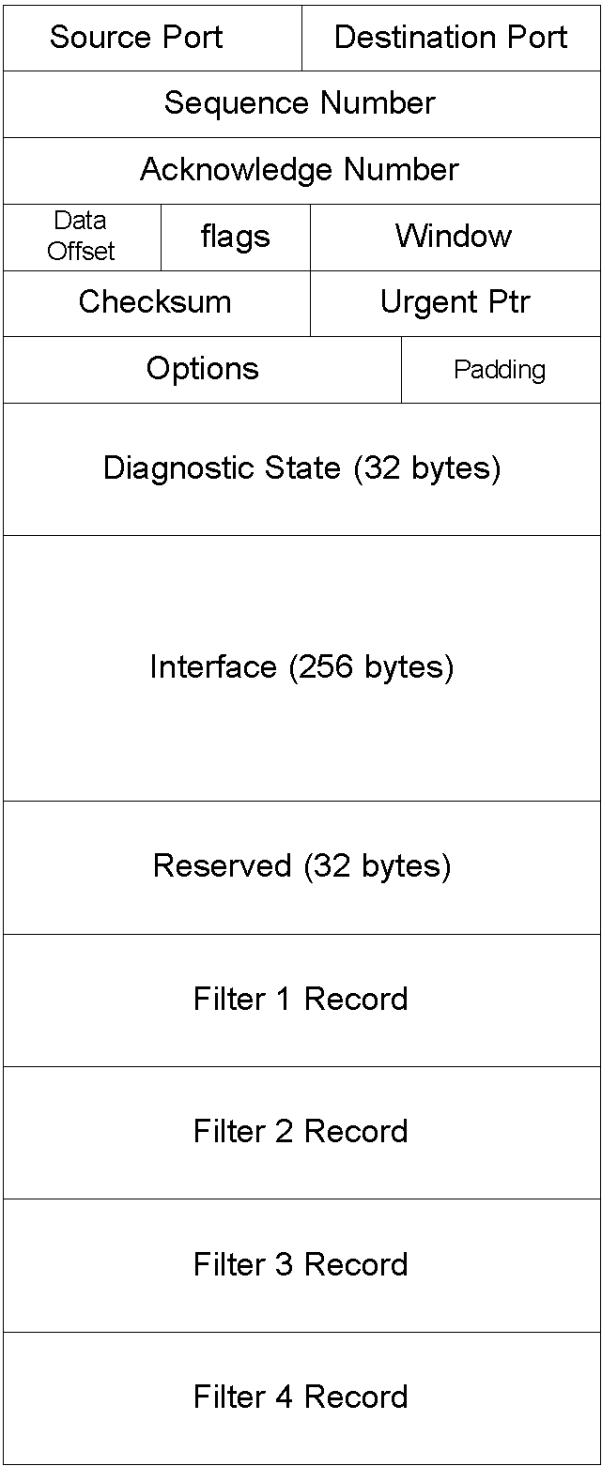

Figure 34 (Passive Usage Monitoring TestResult Message)

The Passive Usage Monitoring TestResult message has a 320 byte header comprised of a 32 byte 'DiagnosticState' field, a 256 byte 'Interface' field, and a 32 byte 'Reserved' field. The DiagnosticState field is filled with the Diagnostic State parameter value of the

test as depicted in Table 10 (Passive Usage Monitoring Test) with the exception that the substring “PUM_” is prepended to Diagnostic State string value. This allows the test server to identify this TestResult message as a Passive Usage Monitoring TestResult message. The ‘Interface’ field is populated as is the Interface parameter in Table 10 (Passive Usage Monitoring Test) with the exception that if the Interface parameter is empty or unassigned (indicating that the BHR WAN Port was monitored), then the string “WAN Port” is placed in this field. Recall that an unassigned or empty value for the Interface parameter is the default value which indicates the BHR WAN port is to be monitored for the Passive Usage Monitoring test. However, if the Interface parameter contains any string value (i.e was configured to indicate a particular interface to monitor), then that value is placed in the ‘Interface’ field. The ‘Interface’ field is followed by a 32 byte ‘Reserved’ field which is in turn followed by four Filter Records. These four Filter Records correspond to the four filters that can be configured per Table 10 (Passive Usage Monitoring Test). Each of these four Filter Records is formatted per Figure 31 (Passive Usage Monitoring Filter Record – Part1) and Figure 32 (Passive Usage Monitoring Filter Record - Part 2).

[H.3] **Theory of Operation:** The Passive Usage Monitoring test is triggered in one of two ways. Either by the user setting the DiagnosticState in the PassiveUsageMonitoring test parameters to “Requested”, or by having the PT bits set to 7 (i.e. PT2 = 1, PT1 = 1, and PT0 = 1), and the CommandExt field set to 0x01, in a TestResponse packet received from the test server (as a result of a TestRequest packet sent to the test server from the device). Once the Passive Usage Monitoring test is triggered it runs for TestDuration seconds, collecting each of UpstreamBytes, UpstreamPackets, DownstreamBytes, and DownstreamPackets measured at the interface (dictated by the Interface parameter) for each one of the four filters that is enabled (i.e. has Enable = TRUE). For example, lets assume we set up and enable four filters (i.e. set Enable = TRUE on .Filter{0}, .Filter{1},. Filter{2}, and .Filter {3}), then a Passive Usage Monitoring test having TestDuration = 3600 and SamplingInterval = 4, will have indexed results, .PassiveUsageMonitoring.Filter{i}.Results.IndividualResults{j}, for each j from 0 to 899 (i.e. it will result in 900 data samples per filter). Each .PassiveUsageMonitoring.Filter{i}.Results.IndividualResults{j} will contain counts for UpstreamBytes, DownstreamBytes, UpstreamPackets, and DownstreamPackets, measured at the BHR physical port defined by the Interface parameter. Now lets assume we have TestDuration = 100 and SamplingInterval = 3. Then in this case the .PassiveUsageMonitoring.Filter{i}.Results.IndividualResults{j} index, j, would range from 0 to 33. That is, it will result on 34 data samples per filter with the last data sample having a length of 1 second (or 1/3 the length of SamplingInterval). Also, the SamplingInterval parameter is ignored if set to zero (or if it is empty or unassigned) so a single date sample is collcted in .PassiveUsageMonitoring.Filter{i}.Results.IndividualResults{0} if the SamplingInterval parameter is set to zero (or it is empty or unassigned). Thus, setting SamplingInterval to zero (i.e. zero, empty or unassigned), or setting SamplingInterval > TestDuration, have

the same effect of resulting a single data sample over the TestDuration period to be collected. However, TestDuration must be > 0, for the test to run.

Appendix I – UDP Throughput Test

The UDP throughput test is included due to it's reduced requirements for BHR device resources in producing high (sending/receiving) bit rates for measuring the throughput of high speed (greater that 500Mbps) service tiers. Unlike the tcp based throughput tests, it does not require the tcp based processing on either the sender or receiver sides including send buffer management, receiver buffer manamgemenr, send sequence number generation, ACKs, triple lost ACK detection, etc … Hence, the UDP throughput test can be used to generate and measure higher speeds than the tcp based throughput tests, though it will also be more intrusive to the the user experience and testing infrastructure as a consequence. As such, it is expected to be used in conjunction with the test admission control features of this specification (see section 3.2.2).

To communicate the start and stop commands and other control functions for this tests a separate tcp channel is opened between the server and client so that the sender can send start and stop messages to the receiver. The test is uni-directional so there will be separate upstream and downstream tests defined. The format of the messages (communicated via the parallel tcp control channel) is:

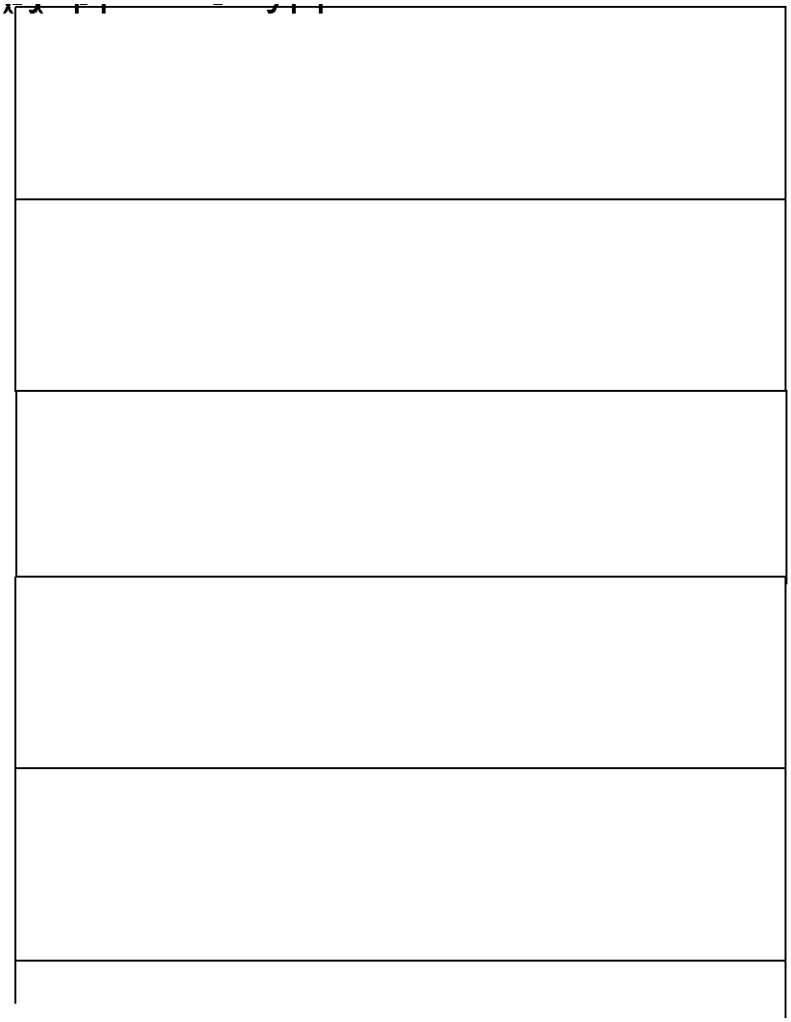

Figure 35 (Control Message on Parallel Control Channel for UDP Throughput Test)

Note that the Control Message is sent from the transmitter (i.e. test server for the download test or BHR for the upload test) to the receiver prior to the UDP test traffic being generated (i.e. prior to the start message). All of the values for the fields in the control message are presented in string format. The (32 byte) Command Type field can currently be set to one of two string values: (1) “Config”, or (2) “Trigger”. The purpose of the “Config” message is to communicate to the receiver what (UDP traffic) bit rates to expect and on what UDP ports to expect it. When Command Type is set to “Config”, then the next (32 byte) field is defined as Tx (Transmit) Speed Per Thread. In this case a string value is provided to communicate the speed of each thread that will be used to transmit UDP traffic from the (UDP throughput) test transmitter to the test receiver. Note the

speed on the line will represent the sum of the speeds for each thread. Also, when the first Command Type field is set to “Config”, the third field (Number of Threads), and fourth field (Max Test Duration) , and fifth field (Transmit UDP ports) are also defined. The Number of Threads field is a string value (e.g. “2” or “3” or “4”, or “5”, etc …) indicating the number of UDP test channels (i.e. UDP ports) used to simultaneously transmit the UDP traffic. The minimum value for the number of threads is “1” and the maximum value of “16”. Since the default number of threads is 1, then an invalid (e.g. “17”) or undefined value translates to a value of “1”. When Command Type is set to “Config”, then the fourth field is the Max Test Duration field. The Max Test Duration is a string value suppled by the test transmitter to the receiver indicating that the test must be completed before this value in seconds. So if Max Test Duation is set to “30”, then the “Stop” message must be sent to the receiver within 30 seconds of the “Start” message. Otherwise the receiver will stop monitoring at 30 seconds and release its test resources (i.e. stop accepting traffic on the UDP test ports). Max Test Duation is included as a protection so that the receiver does not remaing in test mode indefinitely if it never receivers a “Stop” message. Also, when Command Type is set to “Config”, then the 128 byte Transmit UDP ports field is also defined and consist of a list of comma separated UDP ports, each provided as a sting value (e.g. “9000,9001,9002,9003”). This way, the receiver knows what ports to expect the traffic to be generated on for each thread. The number of ports provided in the comma separated list must match the number of threads specified in the previous field. If it doesn’t match, a “TestConfig” error is provided in the DiagnosticState of the test results. The transmitter should wait a sufficient (and configurable) amount of time (e.g. 1 second) between the sending of the “Config” message and the “Start” message which indicates to the receiever that it’s UDP port resources should be open and that it should begin monitoring the UDP traffic on those ports. If Command Type field is set to “Trigger”, then the second field becomes the Start/Stop field and can only be set to one of two string values: (1) “Start”, or (2) “Stop”. When set the “Start”, the transmitter is commanding the receiver to begin its throughput measurement of all packets received on the UDP ports provided previously in the “Config” message. When set to “Stop” it commands the receiver to “Stop” the measurement and end the test at which point the total number of packets and bytes measured on the UDP test ports are tallied and the time between the “Start” and “Stop” time are tallied and used to compute the UDP throughput. Both the the timestamps of the “Start” time and the “Stop” time and the total number of Bytes measured for the duration of the test are passed back to the test server in the TestResult message in a manner analogous to the HTTP throughput test. That is, the timestamp of “Start” message is passed as the “BOMTime” and the timestamp of the “Stop” message is passed back as the “EOMTime”. The total test bytes measured across all of the active test UDP ports as passed back as the TestBytes. The receiver should also measure the TotalBytes received across the line in addition to measuring the TestBytes so that any user activity during the test can also be detected.

Note that the UDP throughput test also supports the incremental test results mode. The incremental test results mode is enabled (for a single threaded or multi-threaded UDP throughput test) whenever DownloadDiagnostics.TimeBasedTestIncrements (unsigned

int – seconds) or UploadDiagnostics.TimeBasedTestIncrements (unsigned int – seconds) is set to a value greater than zero.

If for example, DownloadDiagnostics.TimeBasedTestIncrements is set to 5, and a multi-threaded UDP download test is performed, then starting from the time the "Start" message is received at the receiver, a timestamp is made every 5 seconds and DownloadDiagnostics.TestBytesReceivedUnderFullLoading = DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] is set to the to the value of DownloadDiagnostics.TestBytesReceivedUnderFullLoading during the i-th 5 second interval. Defining timestamp[0] = BOMTime = BOMTime[0] the time the "Start" message is receveived, and timestamp[i] as the i-th timestamp from the "Start" message (which should be BOMTime + i*5 seconds exactly but may vary slightly from the ideal epochs).

As just mentioned, DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] is the i-th sample of DownloadDiagnostics.TestBytesReceivedUnderFullLoading taken at each time stamp. Note that even if the "Stop" message (i.e. EOMTime) were to occur at a non-multiple of 5 seconds (from the "Start" message), then the last timestamp measured at precisely the time the "Stop" message is received (call it timestamp[n]) would still be set equal to the EOMTime. Given these rules, we set DownloadDiagnostics.IncrementalResults{0}.IncBOMTime = timestamp[0] (i.e. the timestamp of the "Start" message), and DownloadDiagnostics.IncrementalResults{i}.IncEOMTime = timestamp[i], and DownloadDiagnostics.IncrementalResults{i}.IncBOMTime = DownloadDiagnostics.IncrementalResults{i-1}.IncEOMTime, for i= 1, 2, …, n. Also, we set DownloadDiagnostics.IncrementalResults{i}.IncTestBytesReceived = DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] - DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i-1] (i.e. the i-th difference in the sampled value of TestBytesReceivedUnderFullLoading as it increments). Similarly, if UploadDiagnostics.TimeBasedTestIncrements is set to 5, the analogous rules apply for the upload case except DownloadDiagnostics.TestBytesReceivedUnderFullLoading[i] is replaced by UploadDiagnostics.TestBytesSentUnderFullLoading[i] and the corresponding upload variables in UploadDiagnostics.IncrementResults{i} substitute for those in DownloadDiagnostics.IncrementResults{i}.

Note that for the UDP throughput test, the values for TcpOpenRequest and TcpOpenResponse are measured by timestamping the SYN and SYN/ACK values are the transmitter as it opens the parallel control channel used to convey the "Config" and "Trigger" (i.e. "Start" and "Stop") messages. So after a mult-threaded test completes (i.e. after a "Stop" message is revieved by the receiver, the BHR opens up a tcp connection to the test server on the TR143 TestResultsTCPPort and sends back a TestResults message per Figures 4 and 5 of section 3.1.6 of this document. Since this is a UDP throughput test, the ROMTime value is set to "0". The BOMTime and EOMTime values are set to the "Start" message and "Stop" message timestamps, respectively. The TestBytesReceived

field is set to the measured value of TestBytesReceivedUnderFullLoading for a downstream test (i.e. from test server to BHR) or TestBytesSent is set to TestBytesSentUnderFullLoading for an upstream test (from BHR to test server). Also TotalByesReceived is set to the total bytes measured across the line for the duration of the test for a a downstream test or the total bytes measured emanating from the BHR for an upstream test. If the test increments where performed during this test (i.e. ???), then the TestResult message is followed by a sequence of TestIncrement messages as defined in Figures 11 and 12 in Appendix A of this document. The j-th TestIncrement message contains DiagnosticState field being populated with the string "TestIncrement j", where j is a number representing the index of the TestIncrement. It also contains incBOMTime[j], incEOMTime[j], incTestBytesReceived and incTotalBytesReceived for a download test or incTestBytesSent and incTotalBytesSent for and upload test.

To support the UDP Throughput test, several parameters are added to the TR143 parameters table.

The additional Tableparameters required for the UDP Throughput test are shown below:

Name	Type	Write	Description	Object Default
.**UDPThroughput**	object	-	This object allows the CPE to be configured to perform the UDP Thro. The PUM test functions by tughput test. For this test a transmitter generates packets over N Concurrent UDP ports and the receiver measured the bytes transferred from the transmitter over a duration of time between a "Start" control message and "Stop" control message.	-
DiagnosticState	string	W	Indicates availability of diagnostic data when the device functions as a UDPThroughput test agent. The value may be one of: "None" "Request" "Requested" "Complete" "Error_Internal" "Error_Other" If the ACS sets the value of this parameter to Requested, the CPE MUST initiate a single PassiveUsageMonitoring test. When writing, the only allowed value is Requested. To ensure the use of the proper test parameters (the writable parameters in this object), the test parameters MUST be set either prior to or at the same time as (in the same SetParameterValues) setting the DiagnosticsState to Requested. When requested, the CPE SHOULD wait until after completion of the communication session with the ACS before starting the diagnostic.	-

			When each test is completed, the value of this parameter MUST be either Complete (if the test completed successfully), or one of the Error values listed above. If the value of this parameter is anything other than Complete, the values of the results parameters for this test are indeterminate. When the diagnostic initiated by the ACS is completed (successfully or not), the CPE MUST establish a new connection to the ACS to allow the ACS to view the results, indicating the Event code "8 DIAGNOSTICS COMPLETE" in the Inform message. After the diagnostic is complete, the value of all result parameters (all read-only parameters in this object) MUST be retained by the CPE until either this diagnostic is run again, or the CPE reboots. After a reboot, if the CPE has not retained the result parameters from the most recent test, it MUST set the value of this parameter to “None”. Modifying any of the writable parameters in this object except for this one MUST result in the value of this parameter being set to “None”. While the test is in progress, modifying any of the writable parameters in this object except for this one MUST result in the test being terminated and the value of this parameter being set to “None”. While the test is in progress, setting this parameter to Requested (and possibly modifying other writable parameters in this object) MUST result in the test being terminated and then restarted using the current values of the test parameters.	
Interface	string(256)	W	IP-layer interface over which the CPE MUST perform the Passive Usage Monitoring test. The content is the full hierarchical parameter name of the interface. The value of this parameter MUST be either a valid interface or an empty string. An attempt to set this parameter to a different value MUST be rejected as an invalid parameter value. If an empty string is specified, the CPE perform this test on the default Interface which is the Broadband Interface (i.e. WAN port) of the device.	-
DSCP	unsignedInt[0:63]	W	When the device functions as a transmitter, the DiffServ code point for marking packets transmitted in the test. The default value SHOULD be zero.	-
EthernetPriority	unsignedInt[0:7]	W	When the device functions as a transmitter, the Ethernet priority code for marking packets transmitted in the test (if applicable). The default value SHOULD be zero.	-
ControlTcpPort	unsignedInt	W	The tcp port that the control channel is performed on,.A tcp connection is opened up on this port when the device functions as the UDPThroughput transmitter.	-
UDPPortList	unsignedInt	W	A comma separated list of UDP ports used for the transansmission of test traffic during the UDP Throughput test,. Note that if more than one UDP port is used for the UDP Thrpoughput test transmission, they must be provide as a comma separated list of continguous port numbers (e.g. “3300, 3301, 3302,3304”). Note that this list only apples when the device functions as a UDP Throughput transmitter and is provided in the “Config” messageof the parallel control channel.	-
TestDuration	unsginedInt (seconds)	W	The duration of time in seconds to perform the UDPThroughput test. This parameter only applies when this devices functions as a UDPThroughput transmitter. It represents the time between when the “Start” message is	-

			sent and the “Stop” message is sent on the control channel.	
TxSpeedPerPort	string	W	When the device functions as a transmitter, the speed in bits per second transmitted across each active UDP port. This is the also the value presented to the receiver via the “Config” message (when the device functions as a transmitter).	-
PacketSize	unsignedInt	W	The size of the packets transmitted when this device functions as a UDP Throughput transmitter.	
TimeBasedTestIncrements	unsignedInt (seconds)	-	When the device functions as a UDPThroughput receiver and TimeBasedIncrements > 0, then hroughput measurements will be partitioned into. So if, for example, TimeBasedTestIncrements = 5 and the difference between the “Start” message and “Stop” message timestamps is 30 seconds, then every 5 seconds the device will measure IncTestBytes, IncBOMTime, and IncEOMTime.). TimeBasedTestIncrements defaults to zero, which implies time segmenting is disabled. TimeBasedTestIncrements is in units of seconds.Note that IncTestBytes represents the aggregate incremental TestBytes measured across all of the active UDP test ports.	-
.**UDPThroughout.UdpPort. Download.Results**	object	-	This object includes the parameters used for measuring either the the bytes sent when the device functions as a transmitter or bytes received when the device functions as a receiver.	-
UDPPortList	unsignedInt	W	The comma separated list of UDP port values used for the UDPThroughput transmission. When the device functions as a receiver, the values are determined from the “Config” message sent by the transmitter. When the device functions as a transmitter this list should match the UDPPorlList parameter value and is the list provided by this device via the “Config” message.	-
BOMTIme	dateTIme	W	When the device functions as a receiver, this value is the timestamp of the time the “Start” message is received over the parallel control channel. When the device functions as a transmitter, this value is the timestamp of the time the “Start” message is sent over the parallel control channel.	-
EOMTime	dateTime	W	When the device functions as a receiver, this value is the timestamp of the time the “Stop” message is received over the parallel control channel. When the device functions as a transmitter, this value is the timestamp of the time the “Stop” message is sent over the parallel control channel.	-
TestRecievedBytesUnderFullLoading	unsignedInt	W	The number of TestBytes received over all of the active UDP ports over the duration (i.e. time between the “Start and “Stop” messages) of the UDPThroughput test.	-
TotalBytesRecievedUnderFullLoading	unsignedInt	W	The total number of bytes measured acrossed the Interface over the duration (i.e. time between the “Start and “Stop” messages) of the UDP Throughput test. This value may differ from the TestBytes measured if other traffic is transferred acrossed the line (i.e. user background activity), during the test.	-
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For the parallel control channel this is the time at which the TCP socket open (SYN) was sent by the transmitter.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For the parallel control channel this is the time at which the TCP SYN/ACK was received by the transmitter.	-
TxSpeedPerPort	string	W	When the device functions as a receiver, this is the value received in the “Config” message.	-

.UDPThroughput.IncrementalRDownloadesults{j}	object	-	The test results collected for the j-th timeslot in a	-
IncTestBytes	unsignedInt	-	When the device functions as a UDPThroughput receiver, IncTestBytes represents the number of bytes measured during the j-th incremental time slot across all of the active UDP Ports.	-
IncBOMTime	dataTime		When the device functions as a receiver, IncBOMTime represents the timestamp of the beginning of the j-th timeslot increment. It is equal to the timestamp of the EOMTime of the (j-1)th timeslot increment.For example, if TimeBasedTestIncrements = 5 and the test is run for 20 seconds then IncBOMTime[1] = BOMTime, IncBOMTIme[2] = IncEOMTime[1] = BOMTime + 5s, IncBOMTime[3] = IncEOMTime [2] = BOMTime + 10s, IncBOMTime[4] = IncEOMTIme[3] = BOMTime + 15s, and IncEOMTime[4] = EOMTtime = BOMTime + 20-s.	-
IncEOMTime	dateTime	-	When the device functions as a UDPThroughput receiver, IncEOMTIme represents the timestamp of the end of the j-th time slot increment. It is equal to the timestamp of the BOMTime of the (J+1)th timeslot increment.	-
.UDPThroughout.UdpPort.Upload.Results	object	-	This object includes the parameters used for measuring either the the bytes sent when the device functions as a transmitter or bytes received when the device functions as a receiver.	-
UDPPortList	unsignedInt	W	The comma separated list of UDP port values used for the UDPThroughput transmission. When the device functions as a receiver, the values are determined from the “Config” message sent by the transmitter. When the device functions as a transmitter this list should match the UDPPorlList parameter value and is the list provided by this device via the “Config” message.	-
BOMTIme	dateTIme	W	When the device functions as a receiver, this value is the timestamp of the time the “Start” message is received over the parallel control channel. When the device functions as a transmitter, this value is the timestamp of the time the “Start” message is sent over the parallel control channel.	-
EOMTime	dateTime	W	When the device functions as a receiver, this value is the timestamp of the time the “Stop” message is received over the parallel control channel. When the device functions as a transmitter, this value is the timestamp of the time the “Stop” message is sent over the parallel control channel.	-
TestBytesSentUnderFullLoading	unsignedInt	W	The number of TestBytes received over all of the active UDP ports over the duration (i.e. time between the “Start and “Stop” messages) of the UDPThroughput test.	-
TotalBytesSentUnderFullLoading	unsignedInt	W	The total number of bytes measured acrossed the Interface over the duration (i.e. time between the “Start and “Stop” messages) of the UDP Throughput test. This value may differ from the TestBytes measured if other traffic is transferred acrossed the line (i.e. user background activity), during the test.	-
TCPOpenRequestTime	dateTime	-	Request time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For the parallel control channel this is the time at which the TCP socket open (SYN) was sent by the transmitter.	-
TCPOpenResponseTime	dateTime	-	Response time in UTC, which MUST be specified to microsecond precision. For example: 2008-04-09T15:01:05.123456 For the parallel control channel this is the time at which the TCP SYN/ACK was received by the transmitter.	-
TxSpeedPerPort	string	W	When the device functions as a transmitter, this is the value that was sent in the “Config” message.	-

.UDPThroughput.IncrementalUploadesults{j}	object	-	The test results collected for the j-th timeslot in a	-
IncTestBytes	unsignedInt	-	When the device functions as a UDPThroughput transmitter, IncTestBytes represents the number of bytes sent during the j-th incremental time slot across all of the active UDP Ports.	-
IncBOMTime	dataTime		When the device functions as a transmitter, IncBOMTime represents the timestamp of the beginning of the j-th timeslot increment. It is equal to the timestamp of the EOMTime of the (j-1)th timeslot increment. For example, if TimeBasedTestIncrements = 5 and the test is run for 20 seconds then, IncBOMTime[1] = BOMTime, IncBOMTIme[2] = IncEOMTime[1] = BOMTime + 5s, IncBOMTime[3] = IncEOMTime [2] = BOMTime + 10s, IncBOMTime[4] = IncEOMTIme[3] = BOMTime + 15s, and IncEOMTime[4] = EOMTtime = BOMTime + 20-s.	-
IncEOMTime	dateTime	-	When the device functions as a UDPThroughput transmitter, IncEOMTIme represents the timestamp of the end of the j-th time slot increment. It is equal to the timestamp of the BOMTime of the (J+1)th timeslot increment.	-

Table 11 (UDP Throughput Test Parameters)

Upon completion of the UDP Throughput test, the BHR will open up a tcp connection over the TestResultsTcpPort. If the device functions as a receiver, the device sends a TestResult message as defined in as defined in Table 4 (TR-143 Test Control Variables – Phase 2) with the Diagnostic State, BOMTime, EOMTime, TestBytesReceived TotalBytesReceieved fields (populated with TestBytesUnderFullLoading, and TotalBytesUnderFullLoading as defined in Table 11), TcpOpenRequestTime, and TcpOpenResponse time fields are not populated The Diagnostic State field is populated with the subtring "_UDPThroughput" appended to the Diagnostic State value as defined in Table 11. So a "Completed" message for the UDP Throughput test would contain the string "Completed_UDPThroughput" in the Diagnostic State field of the Test Result message. It is also required that the BHR place the DownloadURL or UploadURL (as defined inTable 4) in the DownloadURL/UploadURL field of the test result message. When the Upstream UDP Throughput test is performed (i.e. when the BHR functions as a transmitter), then the fields are similarly populated with BOMTime now representing the time the "Start" message is sent from the BHR to the test server and EOMTime representing the time when the "Stop" message is sent from the BHR to the test server. TestBytesUnderFullLoading now represents the TestBytes sent between the "Start" and "Stop" (i.e. BOMTime and EOMTime), and TotalBytesUnderFullLoading now represents the total upstream bytes measured across the line between the "Start" and "Stop" messages. The TcpOpenRequestTime, and TcpOpenResponse time fields are populated with the values of SYN and SYN/ACK times from the opening of the parallel tcp control channel. If both a Downstream and Upstream test are performed (as dictated by both the DTE and UTE bits being set in the test server response to the TestRequest message (i.e. TestResponse message), then the Downstream test is performed first followed by the Upstream test prior to the test result tcp connection being opened over the TestResultsTcpPort. The TestResult message for the downstream test is sent first followed by the TestResult message for the Upstream test, followed by the sending of the TestLocation message. If only a uni-directional Downstream or Upstream

UDPThroughput test performed, then only the corresponding Downstream or Upstream TestResult message is sent followed by the TestLocation mesaage. If test increments were performed, then the TestResult message for the Downstream UDP Throughput test is sent first prior to sending the individual TestIncrement messages (as defined in Figure 11 (TestIncrement packet format - Part 1) and Figure 12 (TestIncrement - Part 2)), followed by the TestResult message for the Upstream UDP Throughput test followed by the individual TestIncrement messages for the upstream test (as defined in Figure 11 (TestIncrement packet format - Part 1) and Figure 12 (TestIncrement - Part 2)), followed by the TestLocation message (as defined in Figure 8 (TestLocation packet format)).

In addition to the testing mechanism, the BHR should implement an admission control policy (or no policy) of whether a UDP high throughput test request can be accepted. BHR should support either of the following or a combination of the following policies, when dediding whether TR-143 UDP high throughput test can be started. BHR design proposal by Greenwave should reflect how this is implemented. The same understand will be shared with Verizon backend system team, throughput testing team, and PLM market team. However, implementing any admission control mechanism should not impact the overall proper operation of the router.

1. BHR can choose to accept the request to start TR-143 UDP high throughput test, knowing that such test may compete with other services on the router for resources such as CPU, memory, network bandwidth etc. Therefore the results may not reach its design goal.
2. Or, the BHR can perform self-check and see if its operating condition can meet the TR-143 test need. Such test could be CPU condition, memory, or network bandwith. E.g., a policy can be WAN BW usage below or above certain threshold.