Abstract

This document extends the capabilities of the NETCONF configuration management protocol in order to standardize mechanisms to perform sets of active tests (i.e., verification) against servers’ running configuration over a period of time to afford the client and server a more robust and resilient configuration management capability. This is of value to commercial enterprise and public networks as well as wireless emergency and military networks. We propose an initial new NETCONF capability, i.e., verify. The verify capability is defined in the context of the verify.yang module listed in the appendix of this document. Associated with the verify operation are various test.yang modules to be defined elsewhere. However, this document defines a set of interoperability requirements on the development of the test.yang modules in order to be controlled through the verify capability.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2011.

Copyright Notice
Copyright (c) 2010 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ......................................................... 3
   1.1. Benefits of This Work ....................................... 6
1.2. Requirements Language ........................................ 7
   1.3. Outline .................................................... 7
2. Definitions ......................................................... 8
3. The Verify Capability ............................................. 8
   3.1. Verify Capability ............................................ 9
      3.1.1. Overview .............................................. 9
      3.1.2. Dependencies ........................................ 10
      3.1.3. Capability Identifier .............................. 10
      3.1.4. New Operations ....................................... 11
         3.1.4.1. <verify> ........................................ 11
         3.1.4.2. <cancel-verify> ................................ 11
         3.1.4.3. <complete-verify> .............................. 11
         3.1.4.4. <verifyStatus> ................................ 12
         3.1.4.5. <verifyComplete> ............................... 12
      3.1.5. Modifications to Existing Operations .................. 12
4. NETCONF Verify Framework ......................................... 12
   4.1. Test Modules ................................................. 12
   4.2. Use Cases .................................................. 12
5. Acknowledgements .................................................. 13
6. IANA Considerations ............................................... 13
7. Security Considerations ......................................... 13
8. References .......................................................... 14
   8.1. Normative References ..................................... 14
   8.2. Informative References .................................... 14
Appendix A. verify.yang Module ...................................... 15
Appendix B. Example ping.yang Module ............................... 21
Authors’ Addresses .................................................. 27
1. Introduction

This document identifies enhancements to NETCONF capabilities to achieve a more robust model of configuration management for future IETF systems. Most network management systems which are required to provide a highly robust network service rely upon some form of out-of-band access for configuration management. This provides an alternative management entry into devices in the event that in-band access is unavailable due to, e.g., mis-configuration. However, not all network deployments can afford the luxury of alternative networks for management access to all networking devices, nor should this be necessary. Examples include Mobile Ad-Hoc Wireless Networks (MANETs) and other forms of Disruption Tolerant Networks (DTNs). All managed networks, as well, would benefit from a more robust and extensive configuration management capability from the IETF, e.g., to provide equivalent network reliability at reduced infrastructure costs. To accomplish this, the NETCONF protocol RFC 4741 [RFC4741] requires extension of capabilities to define and manage active tests and assess success, i.e., Verification, (from both the client and the servers) involving server-side running configuration. This document defines the verify capability within NETCONF.

As an example, we envision a NETCONF client-server interaction model shown in the below figure. Here, the client issues a <commit> with the confirming option. As part of testing prior to issuing the confirming <commit> the client wishes to execute a set of verification tests from the server. It issues the new <verify> operation to manage this aspect of verification testing. The client passes a reference to the server indicating instances of specific pre-configured tests that define the verification test suite. The server executes these as part of the NETCONF <verify> testing process. Simultaneously, the client may also run a set of tests to gain confidence in the proposed configuration changes to the server. Once the server completes its test execution, it indicates success through notification messages. Once the client is comfortable with its own tests and those of the server, it issues the confirming <commit> to the server which forces the server to commit to the proposed configuration change.
Figure 1

This, and other Use Cases, are discussed further in the ‘Framework’ section below.

NETCONF defines the term ‘validation’ as the set of checks performed on proposed configuration code up to the point that the server places it into its running-configuration.
We use the term ‘verification’ as the act of performing active tests against configuration code in the running-configuration on the server. (Note: strictly, verification probably should also cover the act of loading new configuration into the <running> configuration as this may fail, e.g., due to undocumented configuration constraints. However, here we focus on aspects of running active tests to measure network behavior as a form of verification testing.) Verification tests can be executed from either the NETCONF client or the NETCONF server, or from a NETCONF server(a) against running configuration code on a NETCONF server(b), or all combinations.

We define the new :verify capability as a set of stand-alone operations, notifications, and requirements on the definition of test modules for the purpose of managing verification testing on remote servers through standardized mechanisms. This allows for extensible verification testing of configuration across the base of IETF compliant devices. This leads to more resilient configuration management for operators managing multi-vendor networks of devices. This will promote future integrated network management capabilities as opposed to device management capabilities.

A more detailed presentation of the operation of the proposed :verify capability is given in the below figure. Here the client issues the <verify> operation indicating the timeout period, the set of tests which comprise the overall verification test suite, and the nature of the reporting from the server using the associated notification messages. The ‘verifyStatus=true’ indicates that the server should send intermediate status reports following completion of each test set in the suite. At the completion of the entire verification test suite, the server sends the final <verifyComplete> notification to the client.

The <verifyStatus> and <verifyComplete> notifications carry indications of test success or failure based upon pre-configured thresholds and metrics defined within the test module(s) resident on the server. Further, the <verify> operation carries test instance identifiers and switches for various types of reporting, i.e., summary or extended. In total, these place requirements on the definition of interoperable test modules to be developed in support of the :verify capability. We give an example of a ping.test module in Appendix B. Our intent is to define (in other associated documents) a set of connectivity, protocol and transaction test modules to fill out the utility of this verify capability.
1.1. Benefits of This Work

Our objective is to further develop a robust and resilient network configuration capability, building upon the improvements afforded by the NETCONF protocol and its associated modeling language, YANG.
The envisioned benefits of a standardized set of mechanisms and capabilities for verification testing include:

- Minimize faulty configuration and network disconnects,
- Provide for uniform methods for control, execution and reporting of verification testing in multi-vendor networks,
- Improve automation of extensive verification testing,
- Provide opportunity for device modelers to associate/recommend tests tied to specific configuration items, and
- Improve efficiency of coordinated network upgrades.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.3. Outline

In the remainder of this document we next give a set of definitions to be adhered to for the remainder of this discussion. We then provide a new :verify capability which achieves initial aspects of the Robust-NETCONF capabilities. We then examine in the Framework section the relationship of the :verify mechanisms, their relationship to test modules and definitions of metrics and success criteria. The ‘Framework’ section also covers use cases of the :verify capability. Then ‘Acknowledgments’ and ‘IANA Considerations’ are presented. A section on ‘Security Considerations’ is provided concluding the main body of the document.

Various appendices are provided to compliment the text of the main body. Prominent appendices are ‘Appendix A: verify.yang’ which presents the yang module for the :verify capability and ‘Appendix B: ping.yang’ which presents a simple example test module which complies with the requirements of the :verify capability.
2. Definitions

In this section we provide definitions strictly adhered to throughout this document.

The NETCONF specification maintains the following terms:

- NETCONF Client (or client) - this is the management application responsible for the configuration management of network devices.

- NETCONF Server (or server) - this is the device being managed in the network.

We maintain the following distinction between Validation checks and Verification tests:

- Validation checks - checking non-running configuration code against a set of rules, constraints or other requirements. This addresses the total set of checks performed prior to the Server placing the code into its running-configuration.

- Verification tests - measuring behavior of running configuration code against a set of expectations or success criteria. This may be performed through active testing or passive observation and comparison of results against expectations.

- Active measurements perform Verification while rule-based checks perform Validation.

We maintain the following definitions in the descriptions of the :verify capability:

- Verification test (or test set) - a set of identical measurements identified through a single instance identifier and defined in a test module which is pre-configured on the server.

- Verification test suite - the total set of verification tests identified by a set of instance identifiers passed to the server as parameters in the same <verify> message.

3. The Verify Capability

In this section we describe a protocol development which we refer to as the :verify capability. The Yang module for this procedure, i.e., verify.yang, is listed in Appendix A below. An example test module, which complies with the requirements for test modules as spelled out in this document, is the ping.yang module in Appendix B below.
3.1. Verify Capability

3.1.1. Overview

The :verify capability provides a set of standard tools allowing the client to direct verifications tests from remote servers and to collect and uniformly interpret verification test reports related to the success or failure of the tests.

Note: this capability has several prerequisites, including support for <candidate> configuration and notifications.

Additionally, there will be secondary modules for definition of specific verification tests. We present our example in terms of a ping.yang module in Appendix B below. Other test modules will be developed in other documents.

So, a typical client/server interaction would hold:

1. Client sets up the <candidate> configuration on all relevant agents.

2. Client sets up all the relevant test control configuration needed for the verification tests on all relevant agents.

3. Client sends <verify> to all agents with parameters (timeout: seconds, test-template:instance-identifier,verifyStatus:true, extendedStatus:false), i.e.,

   <rpc xmlns="netconf-base" message-id="101">
   <verify xmlns="verify-module">
     <timeout>3600</timeout>
     <test-template xmlns:as="ping-module">
       /at:ping/at:pingEntry[at:pingControlIndex=21]
       /at:ping/at:pingEntry[at:pingControlIndex=42]
       /at:ping/at:pingEntry[at:pingControlIndex=48]
     </test-template>
     <verifyStatus>true</verifyStatus>
     <extendedStatus>false</extendedStatus>
   </verify>
   </rpc>

Figure 3

4. Server returns <ok/>. 
5. The Server runs the tests with the specified (e.g., pingControlEntry) configuration subtree.

6. For each completed test set, the server sends a report in a <verifyStatus> notification.

7. At the completion of the entire verification test suite, the server sends a summary report in a <verifyComplete> notification message.

The client can adjust the nature of the reporting through the 'verifyStatus' and the 'extendedResults' parameters of the &ly;verify> operation. The former determines whether or not <verifyStatus> notifications are sent from the server following the completion of each test set. The later determines whether or not the <verifyStatus> notifications carry raw test data (as defined within the test modules).

Further, the client can decide to immediately cancel all ongoing verification testing by issuing the <cancel-verify> operation. Or the client can decide to cut short the testing by issuing the <complete-verify> operation which instructs the server to complete only the in-progress test set and follow up with <verifyStatus> notification for that completed test set if the client had required this notification message and to wrap up the verification process by sending the <verifyComplete> notification.

3.1.2. Dependencies

The :verify capability requires the existence of test modules resident on servers which comply with the following requirements:

- Must contain a 'list' keyed by a controlIndex which defines a verification test set.

- The 'list' must define an unequivocal means to determine the Boolean result (success or failure) of the specific verification test set. This information is passed in the <verifyStatus> notification.

- The 'list' must define a 'leaf-list' of raw results which may be passed to the client through the <verifyStatus> notification.

3.1.3. Capability Identifier

The :verify capability is identified by the following capability string:
3.1.4. New Operations

3.1.4.1. <verify>

The <verify> operation starts the verification tests on the server. The <verify> operation has four parameters:

- **timeout** - the timeout period associated with the verification test suite. If the timeout expires, the server should complete the in-progress test, send the <verifyStatus> notification for the test if the 'verifyStatus' parameter is set to 'true' and send the <verifyComplete> notification.

- **test-template** - this 'leaf-list' parameter uniquely identifies the verification test set to be performed by the server. Each verification test is identified by an 'instance-identifier' indexing a test definition on a test module resident on the server. Multiple test sets comprise a test suite.

- **verifyStatus** - this parameter is a switch which defaults to 'false' indicating no <verifyStatus> notifications are to be sent from the server to the client. When set to 'true' the server should send a <verifyStatus> notification following the completion of each verification test.

- **extendedResults** - this parameter is a switch which defaults to 'false' indicating no 'anyxml extendedResults' are to be sent in the <verifyStatus> notification. This parameter can only be set to 'true' if the 'verifyStatus' parameter is also set to 'true'. This indicates that the <verifyStatus> notifications should include the raw measurement results carried in the 'anyxml extendResults'.

3.1.4.2. <cancel-verify>

The <cancel-verify> operation immediately cancels a verify test suite in progress on the server. The server terminates in-progress tests immediately and is not required to send any followup notification messages carrying test results.

3.1.4.3. <complete-verify>

The <complete-verify> operation tells the server to complete the in-progress verification test and to send any required followup notifications carrying test results for the current test set and for the test suite.
3.1.4.4.  <verifyStatus>

The <verifyStatus> notification carries the results for each verification test set comprising the entire verification test suite. This notification may also carry raw extended results. This notification is optional and is requested explicitly by the client sending the ‘verifyStatus=true’ parameter in the <verify> operation.

3.1.4.5.  <verifyComplete>

The <verifyComplete> notification is mandatory (unless canceled by the <cancel-verify> operation) and carries a summary result covering the entire verification test suite.

3.1.5.  Modifications to Existing Operations

None.

4.  NETCONF Verify Framework

Note: should rewrite this section to discuss the relationship between :verify capability and supporting test.yang modules. Also discuss the way metrics and thresholds are defined in order to assess test 'pass/failure' decisions. This section should also contain a presentation of the use cases for the :verify capability.

4.1.  Test Modules

NOTE: Need to include a discussion of the test modules, their necessary requirements for interoperability and thoughts on the definition of metrics and thresholds.

4.2.  Use Cases

NOTE: Need to include a discussion of the use cases for the <verify> operation.

These include:

- Used within a <commit> operation with the ‘confirmed’ parameter.
- Used to verify the configuration of server(a) following the re-configuration of server(b).
- Used to verify the <running> configuration prior to copying it into the <startup> configuration.
5. Acknowledgements

The authors wish to thank the many useful suggestions by and discussions with Martin Bjorklund on this capability proposal.

6. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see the update of RFC 2434 [I-D.narten-iana-considerations-rfc2434bis] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

7. Security Considerations

This section presents the required security considerations for all IETF protocols and capabilities. This section was developed following guidelines within RFC 3552 [RFC3552].

This section addresses the security concerns and objectives for the :verify capability within NETCONF. (NOTE: This section is currently TBD.)

Security issues related to the :verify capability should address issues specific to the proposed NETCONF operations. They should also address security issues associated with the development of associated test modules for the purpose of running verification tests. Here is an initial list of potential considerations:

- Verification requires server-side tests that require that packets to be injected into the network for the purpose of measuring some performance characteristics. As such, associated test modules will contain sensitive network and application data; e.g., user IDs and passwords. Further, if security is compromised, this capability could provide a source for denial-of-service, and potential other, attacks.

- The configuration of verification tests may require passing sensitive network information. For this reason, this configuration information should be encrypted prior to transport over the network.
Some test attributes configure username and password information for some application-level protocols as indicated above. Access to these attributes may provide unauthorized use of resources.

Some test attributes configure the size and rate of traffic flows for the purpose of performance measurements. Access to these attributes may exacerbate the use of this capability in denial-of-service attacks. It is recommended that test modules define a maximum packet rate on the device and to indicate this rate. Other objects that control aspects of the test packets related to packet size and rate are will exist in test modules and bounds on these should be set.

Test module objects will exist which set the source and destination addresses on the packet headers. The server should not allow the setting of source addresses on the test packets other than those that are administratively configured onto the server.

8. References

8.1. Normative References


8.2. Informative References


Appendix A. verify.yang Module

In this appendix we list the verify.yang model for use in conjunction with the robust-netconf capabilities.

Note: this capability has several prerequisites, including support for notifications and recommended support for the confirmed-commit capability.

========Contents of "verify.yang"==========

module verify {
    namespace
        "file:///draft-cole-netconf-verify-00.txt";
    prefix "ver";
    organization "IETF";
    contact "[add contact info here].";
    description
        "NETCONF verify procedure."
    revision 2010-01-21 {
        description "Initial version.";
    }
    rpc verify {
        description
            "The verify procedure is started by invoking this operation.

            * A verify procedure is comprised of multiple verification test sets, each indicated by an
              instance-identifier within the ‘test-template leaf-list’ of the <verify> operation. Multiple
              test sets comprise a test suite.

            * the agent will cancel the verify procedure if the <cancel-verify> operation is invoked.

            * the agent will complete the current verification test set and generate the"
<verifyStatus> and <verifyComplete> notifications if the <cancel-verify> operation is invoked.

* the agent will start, monitor, and report the verification test(s) during the time interval after this operation, and before the ‘timeout’ interval has expired.

* the agent will generate the <verifyStatus> notification for each verification test set specified in the ‘test-template leaf-list’, indicating the result of each verification test set.

* the agent will generate the <verifyComplete> notification at the completion of the entire test suite, indicating the final verify procedure status.

* the definition of this capability places requirements on the development of test.yang modules to provide the following set of features:
  - test sets identified by ’instanceId’s,
  - test suites identified by the collection of instanceIds,
  - test suites unambiguously identify: metric and target, Boolean (pass/fail) threshold, (optional) raw data capability.
These requirements are defined in section 3.1.2.

* <verifyStatus> is sent follow each verification test and indicates pass/fail status of test based upon (metric, target, threshold) triplet. It may also carry raw data values from the ‘rawResults’ node carried within the <verifyStatus>’s ‘anyxml extendedStatus’.

";

input {
  leaf timeout {
    description
      "The time interval the agent has to perform the verify operation. If not complete at timeout, then server must issue <verifyStatus> indicating partial test results and that verification tests are being terminated.";
    type uint32;
  }
}
units seconds;
default 600;
}

leaf-list test-template {
description
"Identifies a verification test control entry
or entries for the agent to use for the
verification procedure.

The verification test control entry must conform
to the requirements specified in section 3.1.2,
and the agent must be capable of starting,
monitoring, and reporting the results of
the verification test, as required.

The agent will also generate the
<verifyStatus> notification,
as specified for each verification test st
control entry indicated by this parameter.";
ordered-by user;
type instance-identifier;
min-elements 1;
}

leaf verifyStatus {
description
"A switch indicating the use of the
<verifyStatus notification.  If ‘false’
the client does not want to receive
the <verifyStatus> notification
associated with each verification test set
in the verification test suite. Instead,
it only wants to receive the final
<verifyComplete> notification which
contains a summarized pass/fail result
for the verification test suite.

If ‘true’, then the client is requesting
that the server generates <verifyStatus>
notifications for each verification test set
in the verification test suite.";
type boolean;
default false;
}

leaf extendedResults {
description
"Identifies a verification test control entry
or entries for the agent to use for the
verification procedure.

The verification test control entry must conform
to the requirements specified in section 3.1.2,
and the agent must be capable of starting,
monitoring, and reporting the results of
the verification test, as required.

The agent will also generate the
<verifyStatus> notification,
as specified for each verification test st
control entry indicated by this parameter.";
ordered-by user;
type instance-identifier;
min-elements 1;
}
"A switch indicating that the client is requesting raw test results through the 'anyxml extendedResults'. This defaults to 'false'.

This can only be set to 'true' if the proceeding 'verifyStatus' leaf is set to 'true'. Else, the server should generate an error response to this request."

type boolean;
default false;
progress, then an 'operation-failed' error is generated, and the value 'no-verify' is used for the error-app-tag field.

If the verify procedure in progress cannot be completed for any reason, then an 'operation-failed' error is returned, and the value 'complete-failed' is used in the error-app-tag field.

If any verification test sets associated with this verify test suite are still in progress, they will be canceled by this operation following the completion of the current test set.

If the verify procedure in progress is completed, then the agent will return <ok/>.

notification verifyStatus {
    description
    "Contains the current or final status of a verification test being invoked on behalf of the current verify procedure.";

    list eachTest {
        key "testIdentifier";

        leaf testIdentifier {
            description
            "Indicates which verification test this status report is associated with. This value will identify the same node as specified in a 'test-template' parameter instance provided in the <verify> operation.";
            type instance-identifier;
            mandatory true;
        }

        leaf statusType {
            description
            "Indicates the type of status report that this notification contains.";
            type enumeration {
                enum partial {
description
"Indicates this is a partial status result for this verification test which is still in progress."
}
}

enum final {

description
"Indicates this is the final status result and this verification test which completed or canceled."
}
}

mandatory true;
}

leaf status {

description
"Indicates the NETCONF error-tag value most closely associated with the test status.
The string 'ok' is used to indicate that the pass threshold for the test has been exceeded.";

type string;
reference "RFC 4741bis, Appendix A";
mandatory true;
}

anyxml extendedStatus {

description
"Indicates verification test-specific status data. The requirements for verification tests (section 3.1.2) describes how the semantics of this structure are determined.";

}

notification verifyComplete {

description
"Contains the final status of the current verify test suite."

leaf status {

description
"Indicates the NETCONF error-tag value most closely associated with the test status. The string 'ok' is used to indicate that the pass thresholds were exceeded for
all tests in the verification test suite.

Figure 4

Appendix B. Example ping.yang Module

In this appendix we list an example ping.yang model for use in
conjunction with the :verify capability.

Specifically, the <verify> operation passes the instance-identifiers
in the 'test-template' parameter. Each instance-identifier
identifies a specific ping test. The <verify> operation manages the
identification, execution and reporting of multiple tests within a
single verification test procedure.

```xml
<rpc xmlns="netconf-base" message-id="101">
  <verify xmlns="verify-module">
    <timeout>3600</timeout>
    <test-template xmlns:as="ping-module">
      /at:ping/at:pingEntry[at:pingControlIndex=21]
      /at:ping/at:pingEntry[at:pingControlIndex=42]
      /at:ping/at:pingEntry[at:pingControlIndex=48]
    </test-template>
    <verifyStatus>true</verifyStatus>
    <extendedStatus>false</extendedStatus>
  </verify>
</rpc>
```

Figure 5

========Contents of "ping.yang"=========

module ping {

    namespace "unassigned";

prefix "at";

import ietf-yang-types { prefix yang; }
import ietf-inet-types { prefix inet; }

organization "IETF";

contact
"Andy Bierman
   InterWorking Labs
   EMail: andyb@iwl.com

   Robert G. Cole
   US Army CERDEC
   Email: robert.g.cole@us.army.mil

   Dan Romascanu
   Avaya
   Email:dromasca@avaya.com";

description
"The module for entities implementing the ping test.";

revision 2010-01-27 {
   description "Second revision:
      Added 'pingEntry' list to hold multiple pre-defined test specifications. Added (metric, target, threshold) triplet for pass/fail determination. Added raw data collection and reporting (optional).";
}

leaf test-reference {
   type string;
   config false;
   description "URL for the definition of this test";
}

list pingEntry {
   key "pingControlIndex";
   config true;

   leaf pingControlIndex {

type uint32;
description
   "Identifies the specific control table row of the ping test template to be executed, which represents the verification tests to be performed on the device as part of the verified commit operation.";
}

leaf dstAddr {
    type inet:ip-address;
    description
        "Identifies the destination address in the packet header of the ping message.";
}

leaf srcAddr {
    type inet:ip-address;
    description
        "Identifies the source address in the packet header of the ping message.";
}

leaf spacing {
    type uint32;
    description
        "The number of seconds between sending subsequent ping packets.";
}

leaf startTime {
    type yang:date-and-time;
    config false;
    description
        "The time the first ping packet was sent for the previous test. This is set each time the test is initiated from a client. When this value is reset, the value of the 'result' node is set to 'indeterminant' and the value of the 'received' node is set to zero.";
}

leaf number {
type uint32;
description
"The number of ping packets to be sent."
}

leaf metric {
    type enumeration;
    enum loss {
        description
        "Holds the indication of whether
the transaction was successful (1)
or failed (0).";
    }
    enum delay {
        description
        "Holds the number of milliseconds
for the successful transaction
or '0' if the transaction failed.";
    }
    enum throughput {
        description
        "Holds the measured throughput
in units of bytes/millisecond for
the transaction if successful
or '0' if failed.";
    }
    default "loss";
    description
    "The metric tracked by this specific test.
These values are held on the rawResults
if the specific test indicates storage
of raw data values.";
}

leaf target {
    type uint32;
    description
    "The preformance target for each transaction
measurement. A measured transaction is deemed
successful if its measured 'metric' value
falls within the limits defined by this
'target'. E.g.,
if 'metric = loss', then 'target' must
equal '1' indicating success if repsonse
received.
if 'metric = delay', then responses
received within 'target' milliseconds
are counted as successful.
if ‘metric = throughput’, then responses
received with throughputs greater than
’target’ are counted as successful.

The target value carries the
units defined by the ‘metric’, i.e.,
unitless if ‘metric = loss’,
milliseconds if ‘metric = delay’,
bytes/milliseconds if
’metric = throughput’.

The server counts the number of transaction
measurements that are deemed successful. This
count is compared against ‘threshold’ to
determine overall success or failure of the
test.";
  default "1";
)

leaf threshold {
  type uint32;
  description
    "The threshold value that determines the
     pass/fail status reported to the client
     by this server in the 'verifyStatus'
     notification.";
}

leaf received {
  type uint32;
  config false;
  description
    "The number of successful
     ping transactions received during
     the previous test. This value
     is initialized to zero prior to
     the instantiation of the test
     and is incremented by one for
     each received ping packet. This
     is set each time the test is
     initiated from a client.";
}

leaf result {
  type enumeration {
    enum indeterminant{
      description
        "Set to 'indeterminant' upon
the initiation of a test.

} enum success{
    description
    "Set to 'success' if the
    number of successful pings
    exceeded the 'threshold'."
}

} enum failure{
    description
    "Set to 'failure' if the
    number of successful pings is less
    than or equal to the 'threshold'."
}

} config false;
    description
    "The result of the previous test."
}

leaf rawResultCollection {
    type enumeration;
    enum off {
        description
        "Indicates that the server will
        not store the raw transaction
        measurement values of type indicated
        by metric."
    }
    enum on {
        description
        "Indicates that the server will
        store the raw transaction
        measurement values of type indicated
        by metric. Further, these raw
        measurement values will be passed
        to the client throught 'verifyStatus'
        notification’s 'extendedStatus'
        node."
    }
    config true;
    default "off";
    description
    "A switch to turn ON or OFF the raw
data collection and notification."
}

leaf-list rawResults {
    description
"Holds the raw metric value for each transaction successfully recorded as part of the specific test. The units used for these values conform to the units defined with the 'metric' measured.

Upon completion of this specific test, the server passes this measurement data to the requesting client through the 'verifyStatus' notification’s 'anyxml extendedStatus'.

```plaintext
ordered-by system;
type uint32;
config false;
min-elements 1;
}
```