Network Working Group
Request for Comments: 3430
Category: Experimental

J. Schoenwaelder
TU Braunschweig

December 2002

Simple Network Management Protocol (SNMP) over Transmission Control Protocol (TCP) Transport Mapping

Status of this Memo

Copyright Notice

Abstract

This memo defines a transport mapping for using the Simple Network Management Protocol (SNMP) over TCP. The transport mapping can be used with any version of SNMP. This document extends the transport mappings defined in STD 62, RFC 3417.

Table of Contents

1. Introduction

This memo defines a transport mapping for using the Simple Network Management Protocol (SNMP) [1] over TCP [2]. The transport mapping can be used with any version of SNMP. This document extends the transport mappings defined in STD 62, RFC 3417 [3].

The SNMP over TCP transport mapping is an optional transport mapping. SNMP protocol engines that implement the SNMP over TCP transport mapping MUST also implement the SNMP over UDP transport mapping as defined in STD 62, RFC 3417 [3].

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 BCP 14, RFC 2119 [4].

2. SNMP over TCP

SNMP over TCP is an optional transport mapping. It is primarily defined to support more efficient bulk transfer mechanisms within the SNMP framework [5].

The originator of a request-response transaction chooses the transport protocol for the entire transaction. The transport protocol MUST NOT change during a transaction.

In general, originators of request/response transactions are free to use the transport they assume is the best in a given situation. However, since TCP has a larger footprint on resource usage than UDP, engines using SNMP over TCP may choose to switch back to UDP by refusing new TCP connections whenever necessary (e.g. too many open TCP connections).

When selecting the transport, it is useful to consider how SNMP interacts with TCP acknowledgments and timers. In particular, infrequent SNMP interactions over TCP may lead to additional IP packets carrying acknowledgments for SNMP responses if there is no chance to piggyback them. Furthermore, it is recommended to configure SNMP retransmission timers to fire later when using SNMP over TCP to avoid application specific timeouts before the TCP timers have expired.

2.1 Serialization

Each instance of a message is serialized into a single BER-encoded message, using the algorithm specified in Section 8 of STD 62, RFC 3417 [3]. The BER-encoded message is then sent over a TCP

connection. An SNMP engine MUST NOT interleave SNMP messages within the TCP byte stream.

All the bytes of one SNMP message must be sent before any bytes of a different SNMP message.

It is possible to exchange multiple SNMP request/response pairs over a single (persistent) TCP connection. TCP connections are by default full-duplex and data can travel in both directions at different speeds. It is therefore possible to send multiple SNMP messages to a remote SNMP engine before receiving responses from the same SNMP engine. Note that an SNMP engine is not required to return responses in the same order as it received the requests.

It is possible that the underlying TCP implementation delivers byte sequences that do not align with SNMP message boundaries. A receiving SNMP engine MUST therefore use the length field in the BER-encoded SNMP message to separate multiple requests sent over a single TCP connection (framing). An SNMP engine which looses framing (for example due to ASN.1 parse errors) SHOULD close the TCP connection. The connection initiator will then be responsible for establishing a new TCP connection.

2.2 Well-Known Values

It is RECOMMENDED that administrators configure their SNMP entities containing command responders to listen on TCP port 161 for incoming connections. It is also RECOMMENDED that SNMP entities containing notification receivers be configured to listen on TCP port 162 for connection requests.

SNMP over TCP transport addresses are identified by using the generic TCP transport domain and address definitions provided by RFC 3419 [6], which cover TCP over IPv4 and IPv6.

When an SNMP entity uses the TCP transport mapping, it MUST be capable of accepting and generating messages that are at least 8192 octets in size. Implementation of larger values is encouraged whenever possible.

2.3 Connection Management

The use of TCP connections introduces costs [7]. Connection establishment and teardown cause additional network traffic. Furthermore, maintaining open connections binds resources in the network layer of the underlying operating system.

SNMP over TCP is intended to be used when the size of the transferred data is large since TCP offers flow control and efficient segmentation. The transport of large amounts of management data via SNMP over UDP requires many request/response interactions with small-sized SNMP over UDP messages, which causes latency to increase excessively.

TCP connections are established on behalf of the SNMP applications which initiate a transaction. In particular, command generator applications are responsible for opening TCP connections to command responder applications and notification originator applications are responsible for initiating TCP connections to notification receiver applications, which are selected as described in Section 3 of STD 62, RFC 3413 [8]. If the TCP connection cannot be established, then the transaction is aborted and reported to the application as a timeout error condition. Alternative connection establishment procedures are discussed in Appendix A but are not part of this specification.

All SNMP entities (whether in an agent role or manager role) can close TCP connections at any point in time. This ensures that SNMP entities can control their resource usage and shut down TCP connections that are not used. Note that SNMP engines are not required to process SNMP messages if the incoming half of the TCP connection is closed while the outgoing half remains open.

The processing of any outstanding SNMP requests when both sides of the TCP connection have been closed is implementation dependent. The sending SNMP entity SHOULD therefore not make assumptions about the processing of outstanding SNMP requests once a TCP connection is closed. A timeout error condition SHOULD be signaled for confirmed operations if the TCP connection is closed before a response has been received.

2.4 Reliable Transport versus Confirmed Operations

The transport of SNMP messages over TCP results in a reliable exchange of SNMP messages between SNMP engines. In particular, TCP guarantees (in the absence of security attacks) that the delivered data is not damaged, lost, duplicated, or delivered out of order [2].

The SNMP protocol has been designed to support confirmed as well as unconfirmed operations [9]. The inform-request protocol operation is an example for a confirmed operation while the snmpV2-trap operation is an example for an unconfirmed operation.

There is an important difference between an unconfirmed protocol operation sent over a reliable transport and a confirmed protocol operation. A reliable transport such as TCP only guarantees that delivered data is not damaged, lost, duplicated, or delivered out of order. It does not guarantee that the delivered data was actually processed in any way by the application process. Furthermore, even a reliable transport such as TCP cannot guarantee that data sent to a remote system is eventually delivered on the remote system. Even a graceful close of the TCP connection does not guarantee that the receiving TCP engine has actually delivered all the data to an application process.

With a confirmed SNMP operation, the receiving SNMP engine acknowledges that the data was actually received. Depending on the SNMP protocol operation, a confirmation may indicate that further processing was done. For example, the response to an inform-request protocol operation indicates to the notification originator that the notification passed the transport, the security model and that it was queued for delivery to the notification receiver application. Similarly, the response to a set-request indicates that the data passed the transport, the security model and that the write request was actually processed by the command responder.

A reliable transport is thus only a poor approximation for confirmed operations. Applications that need confirmation of delivery or processing are encouraged to use the confirmed operations, such as the inform-request, rather than using unconfirmed operations, such as snmpV2-trap, over a reliable transport.

3. Security Considerations

It is RECOMMENDED that implementors consider the security features as provided by the SNMPv3 framework in order to provide SNMP security. Specifically, the use of the User-based Security Model STD 62, RFC 3414 [10] and the View-based Access Control Model STD 62, RFC 3415 [11] is RECOMMENDED.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to a MIB is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change) them.

The SNMP over TCP transport mapping does not have any impact on the security mechanisms provided by SNMPv3. However, SNMP over TCP may introduce new vulnerabilities to denial of service attacks (such as TCP syn flooding) that do not exist in this form in other transport mappings.

4. Acknowledgments

This document is the result of discussions within the Network Management Research Group (NMRG) of the Internet Research Task Force[12] (IRTF). Special thanks to Luca Deri, Jean-Philippe Martin-Flatin, Aiko Pras, Ron Sprenkels, and Bert Wijnen for their comments and suggestions.

Additional useful comments have been made by Mike Ayers, Jeff Case, Mike Daniele, David Harrington, Lauren Heintz, Keith McCloghrie, Olivier Miakinen, and Dave Shield.

Luca Deri, Wes Hardaker, Bert Helthuis, and Erik Schoenfelder helped to create prototype implementations. The SNMP over TCP transport mapping is currently supported by the NET-SNMP package[13] and the Linux CMU SNMP package[14].

References

[1] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[2] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[3] Presuhn, R., Ed., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Sprenkels, R. and J. Martin-Flatin, "Bulk Transfers of MIB Data", Simple Times 7(1), March 1999.

[6] Daniele, M. and J. Schoenwaelder, "Textual Conventions for Transport Addresses", RFC 3419, December 2002.

[7] Kastenholz, F., "SNMP Communications Services", RFC 1270, October 1991.

[8] Levi, D., Meyer, P. and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[9] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[10] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[11] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

[12] <http://www.irtf.org/>

[13] <http://net-snmp.sourceforge.net/>

[14] <http://www.gaertner.de/snmp/>

Appendix A. Connection Establishment Alternatives

This memo defines a simple connection establishment scheme where the notification originator or command generator application is responsible for establishing TCP connections to notification receiver or command responder applications. The purpose of this section is to document variations or alternatives of this scheme which have been discussed during the development of this specification. The discussion below focuses on notification originator applications since this is case where people seem to have diverging viewpoints. The discussion below also assumes that the reader is familiar with the SNMPv3 notification forwarding model as defined in STD 62, RFC 3413 [8].

The variations that have been discussed are basically driven by the idea of providing fallback mechanisms in cases where TCP connection establishment from the notification originator to the notification receiver fails. The approach specified in this memo simply drops notifications if the TCP connection cannot be established. This implies that notification originators which need reliable notification delivery must implement a local notification log in order to keep a history of notifications that could not be delivered.

Another option is to deliver notifications via UDP in case TCP connection establishment fails. This might require augmenting the snmpTargetTable with columns that provide information about the alternate UDP transport domain and address. In general, this approach only helps to deliver notifications in cases where the notification receiver is unable to accept more TCP connections. In other fault scenarios (e.g. routing problems in the network), the UDP packet would have no or only marginally better chances to reach the notification receiver. This implies that notification originators which need reliable notification delivery still need to implement a local notification log in order to keep a history of notifications in case the UDP packets do not reach the destination.

A generalization of this approach leads to the idea of a sparse augmentation of the snmpTargetTable which lists alternate fallback transport endpoints of arbitrary transport domains. Multiple fallbacks may be possible by using a tag list approach. This provides a generic transport independent fallback mechanism which is independent of the TCP transport mapping defined in this memo.

Another alternative is to make the notification originator responsible for retrying connection establishment. This could be accomplished by augmenting the snmpTargetTable with additional columns that specify retry counts and timeouts or by adapting the existing snmpTargetAddrTimeout and snmpTargetAddrRetryCount columns in the snmpTargetTable. But even this approach requires a local notification log in order to handle situations where all retries have failed.

A fundamentally different approach is to make the notification receiver responsible for establishing the TCP connection to the notification originator. This approach has the advantage that the notification originator does not necessarily need a list of pre-configured notification receiver transport addresses. The current notification forwarding model however relies on the snmpTargetTable to identify notification targets. So the question comes up whether (a) new entries are added to the snmpTargetTable when a connection is established or whether (b) connections are only accepted if they match pre-configured snmpTargetTable entries. Note that the target selection logic relies on a tag list which can not be reasonably populated when a connection is accepted. So only option (b) seems to be compliant with the current notification forwarding logic. Another issue to consider is the vulnerability to denial of service attacks. A notification originator can be easily attacked by syn-flooding attacks if it listens for incoming TCP connections. Finally, in order to let notification originator and notification receiver applications coexist easily on a single system, it would be necessary to assign new default port numbers on which notification originators listen for incoming TCP connections.

Author’s Address

Juergen Schoenwaelder
TU Braunschweig
Bueltenweg 74/75
38106 Braunschweig
Germany
Phone: +49 531 391-3283
EMail: schoenw@ibr.cs.tu-bs.de

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.