Performance Analysis of SNMP over SSH

Vladislav Marinov and Jürgen Schönwälder

Computer Science
International University Bremen
Campus Ring 1, 28759 Bremen, Germany
{v.marinov, j.schoenwaelder}@iu-bremen.de

Abstract. There have been several attempts in the past to secure the Simple Network Management Protocol (SNMP). Version 3 of the SNMP protocol introduced a User-based Security Model (USM) which comes with its own user and key-management infrastructure. However, many operators are reluctant to introduce a new user and key management infrastructure just to secure SNMP. This paper describes how the Secure Shell (SSH) protocol can be used to secure SNMP and it provides a performance analysis of a prototype implementation which compares the performance of SNMP over SSH with other secure and insecure versions of SNMP.

1 Introduction

Network devices maintain large amounts of management data. Management data can provide insights as to how the network is performing or which abnormal events have been observed. Moreover, management data can be used to understand how a device is configured and to change the configuration of that device. The Simple Network Management Protocol (SNMP) [1] allows both for management data to be collected remotely from devices and for devices to be configured remotely. It was first published in August 1988 and since then it has been widely used in network management.

There was no security implemented in SNMP version one (SNMPv1) and the attempts to add security to SNMP version two (SNMPv2) lead to failure as well. Version 3 of the Simple Network Management Protocol (SNMPv3) added security to the previous versions of the protocol by introducing a User-based Security Model (USM) [2]. The USM was designed to be independent of other existing security infrastructures, to ensure it could function when third party authentication services were not available, such as in a broken network. As a result, USM utilizes a separate user and key management infrastructure.

Network operators have reported that deploying another user and key management infrastructure introduces significant costs and hence the USM design is actually a reason for not deploying SNMPv3. To address this issues, a new security model is currently being defined by the Integrated Security Model for SNMP (ISMS) working group of the Internet Engineering Task Force (IETF) which leverages the Secure Shell (SSH) [3] protocol. It is designed to meet the security and operational needs of network administrators, maximize usability in operational environments to achieve high deployment success and at the same time minimize implementation and deployment costs to minimize the time until deployment is possible.

This paper describes the SSH security model for SNMP and provides a performance evaluation of a prototype implementation. It is structured as follows: Section 2 describes the extensions of the SNMP architecture that are needed to support security models where security services such as authentication and encryption are provided by the message transport rather than the SNMP protocol itself. Section 3 introduces the SSH security model for SNMP. A prototype implementation of SNMP over SSH is described in Section 4 and some performance figures are presented in Section 5. Section 6 discusses related work before we conclude our paper in Section 7.

2 Extensions of the SNMP Architecture

The SNMP architecture [4] was designed to be modular in order to support future protocol extensions such as additional security models. The architecture defines several subsystems and interfaces between subsystems that should remain unchanged when subsystems are extended. The goal was to reduce side effects that can occur without such an architectural framework when the protocol is extended.

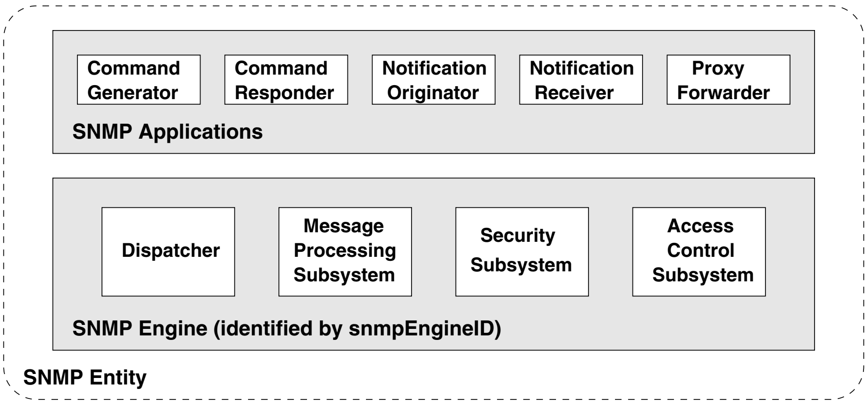

Fig. 1. Structure of an SNMP entity according to the SNMPv3 architecture

According to the SNMP architecture, an SNMP engine consists of a message processing subsystems, a security subsystem, an access control subsystem, and a single dispatcher (Fig. 1). Each subsystem can contain multiple concrete models that implement the services provided by that subsystem. The interfaces between subsystems are defined as Abstract Service Interfaces (ASIs). The dispatcher is a special component which controls the data flow from the underlying transports through the SNMP engine and up to the SNMP applications[1].

As of today, most SNMPv3 implementations support three message processing models for SNMPv1, SNMPv2c, and SNMPv3 and two security models, namely the User-based Security Model (USM) (used by the SNMPv3 message processing model) and the Community-based Security Model (CSM) (used by the SNMPv1 and SNMPv2c message processing models). There is only a single View-based Access Control Model (VACM) so far.

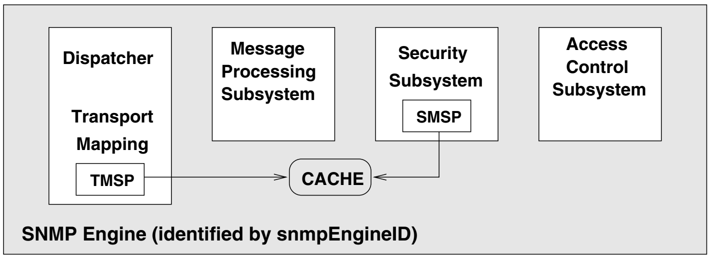

Fig. 2. Structure of an SNMP engine that supports transport mapping security models. The Transport Mapping Security Processor (TMSP) and the Security Model Security Processor (SMSP) communicate via a shared cache.

The design of the SNMP architecture assumes that security services (authentication, data integrity checking, encryption) are provided as part of the SNMP message processing. If, however, the security services are provided by the transport over which SNMP messages are exchanged, the architecture does need some extensions. The approach followed by the ISMS working group of the IETF [5] is to split a transport mapping security model (TMSM) into two parts (Fig. 2):

– The *Transport Mapping Security Processor* (TMSP) is the portion that is part of the message transport and performs the actual security processing.
– The *Security Model Security Processor* (SMSP) is the portion that realizes the appropriate security model required by the SNMPv3 architecture. In order to provide the required services, it has to interact with the TMSP.

The TMSP and the SMSP need to exchange information (e.g., the name name of the authenticated SSH user). While this exchange can be realized in different ways, the simplest and most efficient scheme is to establish a cache which maintains relevant information and to pass a handle between the TMSP and SMSP by extending the ASIs.

[1] The SNMP architecture should actually have a separate transport subsystem with proper ASIs to cleanly model the fact that SNMP supports multiple transport models. The introduction of a transport subsystem has recently been proposed to the ISMS working group.

Transport security protocols are typically session-based. They usually have a session establishment phase where a session key and some shared state is established followed by the secured data exchange. This is very different from the message-based approach used by USM where all security information is carried in every single message exchanged between two SNMP engines and there is no notion of a session or a session key.

3 SSH Security Model for SNMP

The Secure Shell (SSH) protocol [3] is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major components:

– The *Transport Layer Protocol* provides server authentication, confidentiality, and integrity. It may optionally also provide compression. The transport layer protocol typically runs over a TCP/IP connection, but might also be used on top of any other reliable data stream. It uses public-key cryptography to authenticate the server to the client and to establish a secure connection which then uses a session key and a symmetric encryption algorithm to protect the connection.
– The *User Authentication Protocol* authenticates the client-side user to the server. It runs over the transport layer protocol. SSH supports several different user authentication mechanisms such as password authentication, public-key authentication, and keyboard-interactive authentication (which supports challenge-response authentication mechanisms).
– The *Connection Protocol* multiplexes the encrypted connection into several logical channels. It runs over the transport layer protocol after successful completion of the user authentication protocol. Every channel has its own credit-based flow control state in order to deal with situations where channels are connected to applications with different speeds.

Note that SSH authentication is usually asymmetric: An SSH server authenticates against an SSH client using host credentials (host keys) while the user authenticates against the SSH server using user credentials (user keys or passwords).

The SSH Security Model (SSHSM) for SNMP [6] is an instantiation of a TMSM which uses SSH, a protocol already widely deployed to secure access to command line interfaces on network elements. The specification details the elements of procedure for the TMSP and the SMSP portions of the SSHSM. It also deals with details such as `engineID` discovery and the handling of notifications. Notification delivery is not straight forward due to the asymmetric authentication provided by SSH and the requirement to exercise access control in a consistent way for read, write, and notify access. The details are still being discussed in the ISMS working group of the IETF at the time of this writing.

With the SSHSM, no security parameters are conveyed in SNMPv3 messages. Accordingly, the `msgSecurityParameters` field of SNMPv3/SSH messages carries a zero length octet string and the implementation of the security model portion of the SSHSM simply retrieves the necessary information by accessing a cache which is shared between the transport mapping porting and the security model portion of the SSHSM.

4 Implementation

The prototype implementation of SNMP over SSH developed at IUB is an extension of the widely used open source `Net-SNMP` SNMP implementation. For the SSH protocol, the `libssh` library was used, an open source C implementation of SSH. The `libssh` library contains all functions required for manipulating a client-side SSH connection and an experimental set of functions for manipulating a server-side SSH connection.

The implementation does not implement the SSHSM as it is currently discussed in the IETF since the details of the SSHSM were not worked out when the implementation work was done. The prototype only supports the TMSP part of SSHSM plus a slight modification of the Community-based Security Model (CSM) which passes the authenticated SSH user identity as the security name to the access control subsystem. This basically gives us SNMPv1/SSH and SNMPv2c/SSH while the ISMS working group defines SNMPv3/SSH.

The implementation itself consists of a new transport module which is in the order of 1200 lines of C code. The `Net-SNMP` internal API for adding transports worked well and did not require any changes. The fact that `Net-SNMP` already supports stream transports was convenient. For password authentication, the prototype calls the Linux Pluggable Authentication Modules (PAM) [7] library to make it runtime configurable how passwords are checked.

Most of the development time was spend on optimizing the performance of the implementation since the overall latency initially was surprisingly high. In order to optimize the performance of the SSH transport domain, we investigated the influence of TCP's Nagle algorithm as well as the windowing mechanism of the SSH protocol.

4.1 TCP Nagle Interactions

During our initial measurements, we observed that the execution of a `snmpget` operation over the SSH transport domain required approximately $800ms$. This surprisingly large delay was introduced by TCP's Nagle algorithm which essentially delays the sending of a TCP segment until either a segment has been filled or the previous segment has been acknowledged. We therefore disabled the Nagle algorithm by setting the `TCP_NODELAY` flag on the agent and on the manager side of the connection. This lead to a significant improvement in the performance of the SSH transport domain as the time required for the execution of a `snmpget` operation went down to $56.5ms$. We further modified the `libssh` library to disable the Nagle algorithm immediately after establishing the TCP connection between the agent and the manager and before any SSH exchanges. This further reduced the time required for a `snmpget` operation to $16.17ms$ on our fast machines.

4.2 SSH Window Adjustments

The SSH windowing mechanism is used to specify how much data the remote party can send before it must wait for the window to be adjusted. In the OpenSSH implementation such window adjustment messages are only exchanged periodically. During our initial observations we noticed that each message exchanged between the agent and the manager was followed by a window adjustment message. These additional messages introduced significant bandwidth overhead as well as latency overhead for long sessions. As a result the SSH transport domain performed worse than the USM transport domain with respect to latency and bandwidth. In order to optimize the performance, we modified the `libssh` library to send window adjustment messages only when necessary. This improvement lead to better bandwidth and latency performance of the SSH transport domain when compared to the USM transport domain as explained below.

5 Performance Analysis

The performance of our SNMP over SSH prototype has been evaluated by comparing it against SNMPv3/USM with authentication and privacy enabled, running over both TCP and UDP. In addition, to establish a baseline, the performance of plain SNMPv2c over both TCP and UDP was measured. The following sections first describe the experimental setup and then discuss the session establishment overhead and the performance for walks of different sizes without and with packet loss. Finally, the bandwidth used by the different SNMP transports is compared and the memory requirements for keeping open SSH sessions on a command responder is discussed.

5.1 Experimental Setup

The experiments were performed on three Debian GNU/Linux machines (see Table 1). The machines were connected via a switched Gigabit Ethernet with sufficient capacity.

Table 1. Machines used during the measurements

Name	CPUs	RAM	Ethernet	Kernel
`meat`	2 Xeon 3 GHz	2 GB	1 Gbps	2.6.16.14
`veggie`	2 Xeon 3 GHz	1 GB	1 Gbps	2.6.12.6
`turtle`	1 Ultra Sparc IIi	128 MB	100 Mbps	2.6.16.14

The SNMP command responder was running on `meat` and `turtle` for the measurements without packet loss and `veggie` was acting as a command generator. Under the condition of packet loss, the command responder was running on `turtle` while the command generator was running on `meat`. For the

SNMP/USM measurements, we used the authentication plus privacy security level with HMAC-MD5 as the authentication algorithm and AES-128 as the encryption algorithm. The `libssh` library was configured to also use the HMAC-MD5 authentication algorithm and the AES-128 algorithm for encryption and null compression.

We instrumented the `snmpget`, `snmpwalk` and `snmpbulkwalk` programs (part of the `Net-SNMP` package) to measure the latency by calling `gettimeofday()` before opening a session (but after parsing of MIB files) and after closing the session and computing the time difference. The output was directed to `/dev/null` and each experiment was repeated 100 times. The `tcpdump` tool was used to calculate the number of bytes exchanged and the `pmap` tool was used to measure the memory sizes of the processes. Packet loss was simulated by using the `netem` network emulation queuing discipline of the Linux kernel on `meat` and `turtle`.

The object identifier (OID) used for `snmpwalk` and `snmpbulkwalk` measurements was the interface table `ifTable` [8]. The object identifier used for `snmpget` measurements was the scalar `sysDescr` [9]. The number of rows in the `ifTable` was manipulated by creating virtual LAN (VLAN) interfaces.

5.2 Session Establishment

Table 2 shows the result of performing `snmpget` requests on the scalar `sysDescr` using different transports. There is a significant cost associated with the establishment of SSH sessions. This is, however, not surprising since the SSH protocol establishes a session key using a Diffie-Hellman key exchange (using public-key cryptography) before the user authentication protocol is executed and the SSH channel is established. Since the cryptographic operations are CPU bound, the session establishment times increases significantly on our slow test machine.

Table 2. Performance of `snmpget` requests (`sysDescr.0`)

Protocol	Time (`meat`)	Time (`turtle`)	Data	Packets
SNMPv2c/UDP	1.03 ms	0.70 ms	232 byte	2
SNMPv2c/TCP	1.13 ms	1.00 ms	824 byte	10
SNMPv3/USM/UDP	1.97 ms	2.28 ms	668 byte	4
SNMPv3/USM/TCP	2.03 ms	3.03 ms	1312 byte	12
SNMPv2c/SSH	16.17 ms	91.62 ms	4388 byte	32

Table 2 also shows that there is an clear difference in the amount of data (total size of the Ethernet frames) and the number of IP packets exchanged between UDP and TCP transports. While this overhead is usually not a big issue on a well functioning local area network, it might be an issue in networks with large delays or high packet loss rates.

5.3 Latency Without Packet Loss

Figure 3 shows the latency for the retrieval of the `ifTable` [8] with different sizes using `snmpwalk`. The difference between the transports UDP and TCP seems marginal compared to the difference caused by enabling authentication and privacy. The performance of the SSH transport is interesting. Initially, the costs of establishing an SSH channel with a new session key cause the SSH transport to perform worse than SNMPv3/USM. However, there is a break-even-point after ≈ 500 SNMP interactions where the SSH transport becomes more efficient than SNMPv3/USM. This observation seems to be independent of the speed of the machine hosting the command responder.

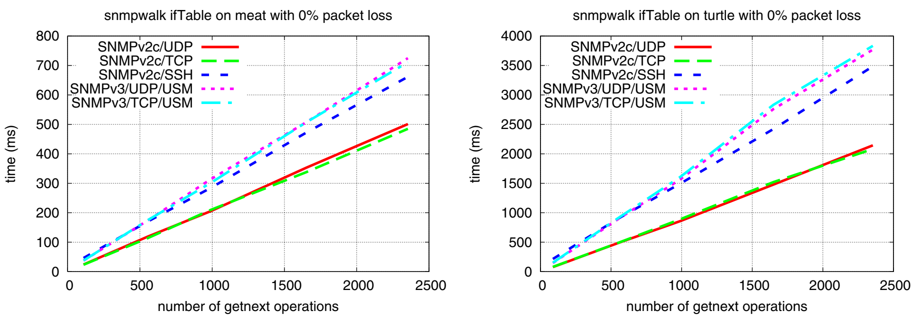

Fig. 3. Latency comparison of SNMP `getnext` walks (`ifTable`) without packet loss with a command responder on a fast machine (left plot) and on a slow machine (right plot)

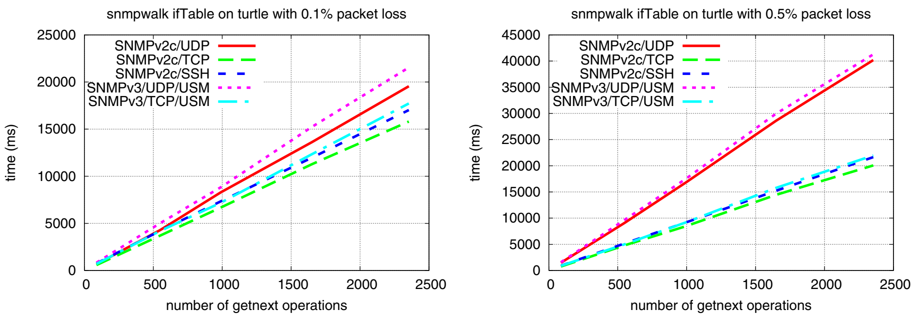

Fig. 4. Latency comparison of SNMP `getnext` walks (`ifTable`) under packet loss with a command responder on a slow machine with different packet loss probabilities

5.4 Latency with Packet Loss

Figure 4 shows the latency of the same `snmpwalks` on the `ifTable` with 0.1% and 0.5% packet loss using the slow command responder running on `turtle`. The surprising result is that the TCP-based transports all clearly outperform the UDP-based transports. It turns out that `Net-SNMP` has a very simple retransmission scheme with a default timeout of 1 second and 5 retries. TCP reacts much faster to lost segments in our experimental setup and this explains the bad performance of the UDP transports in the plots of Figure 4. Note that this result cannot be generalized since other SNMP implementations may have other retransmission schemes. However, simple statements that UDP transports outperform TCP transports in lossy networks are questionable as long as the application layer retransmission scheme is not spelled out.

5.5 Bandwidth

Figure 5 shows the amount of data exchanged during the `snmpwalk` experiments without packet loss. It can be seen that the amount of data exchanged increases when switching from UDP to TCP due to larger TCP headers. Furthermore, SNMPv2c/SSH requires less bandwidth than SNMPv3/USM/UDP and SNMPv3/USM/TCP. Even though the SSH connection protocol adds a tiny header, it seems that this header is significantly shorter than the space needed for the SNMPv3/USM header.

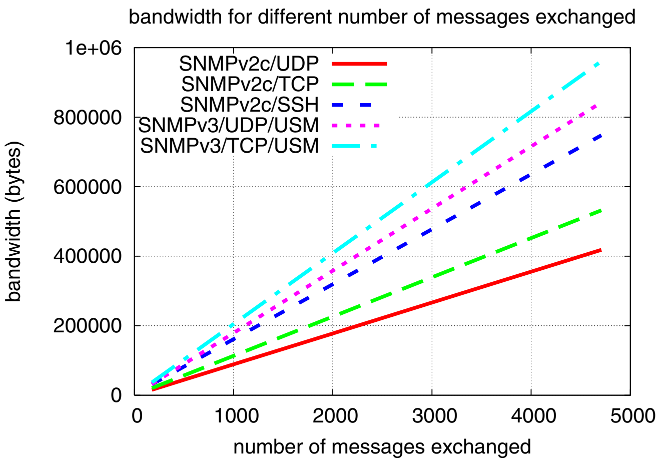

Fig. 5. Bandwidth used for different numbers of exchanged SNMP messages

Table 3 indicates that the change from SNMPv2c to SNMPv3/USM costs approximately 90 bytes of overhead per packet while the SNMPv2/SSH prototype adds approximately 40 bytes of overhead per packet to the SNMPv2c/TCP transport. While the SNMPv3 message header is slightly larger than the SNMPv2c header we have used in our implementation, we believe that also an SNMPv3/SSH implementation will consume less bandwidth than SNMPv3/USM when many packets are exchanged. As can be seen from the total number of packets, the piggy-backing of TCP ACKs worked nicely for all TCP transports.

Table 3. Packet sizes for the `snmpwalk` with 2354 `getnext` operations. The 'range' column shows the dominating packet size range and the 'packets' column the number of packets in that range; the column 'total packets' indicates the total number of packets.

Protocol	Range	Packets	Total Packets
SNMPv2c/UDP	80-100	4710	4710
SNMPv2c/TCP	110-130	4709	4712
SNMPv3/USM/UDP	170-190	4710	4712
SNMPv3/USM/TCP	200-220	4709	4714
SNMPv2c/SSH	150-170	4711	4742

5.6 Memory Usage

Figure 6 shows the amount of virtual memory used by the command responder process for an increasing number of concurrently open sessions. Initially, the process needed 9312 KByte of memory. The memory consumption grows linearly with the number of concurrently open sessions. The measured memory overhead per session is approximately 7 KByte. Note that the sessions were only established but not used during these measurements. Some additional memory might be dynamically allocated by the SSH library once data exchanges take place.

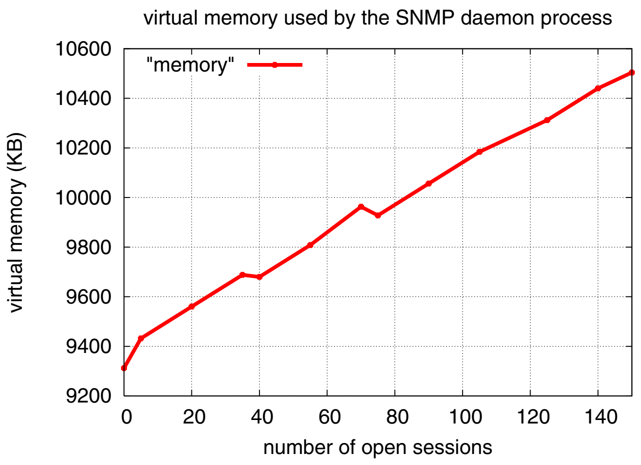

Fig. 6. Virtual memory consumed by the command responder process for an increasing number of open sessions

6 Related Work

SSH is already widely used to secure the access to command line interfaces on network elements. Accordingly, SSH credentials (keys, passwords) are readily available in many environments. This availability of credentials in many operational networks has been the main motivation for considering SSH as a secure transport for SNMP.

The obvious alternative to SSH is the widely used Transport Layer Security (TLS) protocol [10]. An implementation of SNMP over TLS has been analyzed in [11]. Although the authors used a very different setup, some key results are similar to the results reported in this paper. The authors of [11] also observed that SNMP over TLS is more efficient in terms of latency than SNMPv3/USM for longer sessions.

The short-coming of the work done on SNMP over TLS back in 2001 was that architectural questions were not considered and it thus remained unclear how model independent values such as the SNMP security name or security level are determined and passed around. These architectural questions have meanwhile been addressed in the IETF as described in [5] and summarized in Section 2.

The work reported in [12] compares the costs of using SNMPv3/USM with different security levels with SNMPv1 and SNMPv2c. While some of the measurements reported in this paper are comparable with results reported in [12] (e.g., the network capacity consumed for `snmpget`), the results are not identical and differ slightly. In particular, the overhead of SNMPv2/USM with authentication and privacy seems to be generally higher compared to SNMPv1/SNMPv2c in our measurements. This may be explained by the fact that the SNMP engines used in both studies are different as well as the operating system and hardware platform used. In addition, different MIB objects were fetched in both experiments: This study uses the `sysDescr` scalar, a `DisplayString` of ≈ 60 characters, while an 4 byte `IpAddress` object was used in [12].

7 Conclusions

The ISMS working group of the IETF is working on new security models for SNMP which leverage a secure transport. One crucial question is how the performance of this new approach compares to the existing security solution for SNMP (SNMPv3/USM) and to the still widely deployed insecure versions of SNMP (SNMPv1/SNMPv2c).

The measurements presented in this paper try to give answers to some of these questions. In particular, we quantified the session establishment overhead for SNMP over SSH. For simple one-shot SNMP requests, SSH seems to be a rather costly solution since the costs for establishing a session and associated session keys is significant. For sessions that carry multiple SNMP interactions (e.g., table walks), the costs for the initial session setup are amortized and there is a break-even-point where SNMP over SSH starts to become more efficient than SNMPv3/USM with authentication and privacy enabled.

The answer to the question whether SNMP over SSH is a viable alternative to SNMPv3/USM therefore depends on the SNMP usage pattern and the typical session length. While SNMP traditionally has no concept of a session, it is possible to approximate session life times by analyzing SNMP traffic traces. Work is underway in the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) to collect SNMP traffic traces from different operational networks [13]. These traces are expected to give insights in which environments SNMP over SSH is likely to be a viable alternative.

Acknowledgments

The work reported in this paper was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

References

1. J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, December 2002.
2. U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 3414, December 2002.
3. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006.
4. D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, December 2002.
5. D. Harrington and J. Schönwälder. Transport Mapping Security Model (TMSM) Architectural Extension for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-ietf-isms-tmsm-03.txt>, June 2006.
6. D. Harrington and J. Salowey. Secure Shell Security Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-secshell-02.txt>, June 2006.
7. A. G. Morgan. The Linux-PAM Application Developers' Guide. Technical report, November 1999.
8. K. McCloghrie and F. Kastenholz. The Interfaces Group MIB. RFC 2863, June 2000.
9. R. Presuhn. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). RFC 3418, December 2002.
10. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, 2006.
11. X. Du, M. Shayman, and M. Rozenblit. Implementation and Performance Analysis of SNMP on a TLS/TCP Base. In *Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management*, pages 453–466, Seattle, May 2001.
12. A. Corrente and L. Tura. Security Performance Analysis of SNMPv3 with Respect to SNMPv2c. In *Proc. 2004 IEEE/IFIP Network Operations and Management Symposium*, pages 729–742, Seoul, April 2004.
13. J. Schönwälder. SNMP Traffic Measurements. Internet Draft (work in progress) <draft-irtf-nmrg-snmp-measure-00.txt>, May 2006.