Implementation and Performance Analysis of SNMP on a TLS/TCP Base[*]

X. Du, M. Shayman
Department of Electrical and Computer Eng.
University of Maryland
College Park, MD 20742
USA
{dxj, shayman}@glue.umd.edu

M. Rozenblit
TeraBurst Networks Inc.
1965 Broadway, Apt 14G
New York, NY 10023
USA
mrozenblit@teraburst.com

Abstract

There is recent interest in exploring SNMP/TCP in addition to the current use of SNMP/UDP due to performance benefits for bulk transfer as well as to simplify management applications. If SNMP is implemented over TCP, then TLS is a natural choice for security. However, it must be demonstrated that the additional overhead associated with TLS is not excessive. We show this by implementing SNMP on a TLS/TCP base and measuring its performance experimentally. The results indicate that the overhead is not excessive; consequently SNMP/TLS/TCP appears to be a viable option for network management. Also our tests show that SNMP

Keywords

SNMP, TLS, SNMP/TLS/TCP, overhead, integrity protection, privacy protection

1. Introduction

As a highly effective set of automated tools for managing today's diverse, multivendor systems, Simple Network Management Protocol (SNMP), along with the Remote Network Monitoring (RMON) technology, is recognized as the de-facto standard in the field of network management for IP-based networks. It is a popular protocol used in managing computers, peripherals, and data network devices.

SNMP was initially specified in the late 1980s and quickly became the standard means for multivendor network management. However, SNMP was too limited to meet all critical network management needs. Three enhancements have solidified the role of SNMP as the indispensable network management tool. First the RMON specification, which is built on SNMP, was released in 1991. RMON was revised in 1995, and an enhancement to RMON, known as RMON2, was

[*] Research partially supported by the Laboratory for Telecommunications Science under contract MDA90499C2521.

issued in 1997. RMON defines a MIB for managing remote LANs. Second, an enhanced version of SNMP, known as SNMPv2c, was released in 1993 and revised in 1995. SNMPv2c provides more functionality and greater efficiency than in the original version of SNMP. Finally, SNMPv3 was issued in 1998. SNMPv3 defines an overall framework for present and future versions of SNMP and adds security features to SNMP.

All the three SNMP versions (SNMPv1, SNMPv2c and SNMPv3) are normally implemented using the User Datagram Protocol (UDP) for transport (layer 4). When large amounts of data need to be transferred, they must be transported using small-sized SNMP over UDP messages which result in excessive latency [1,5]. Transporting SNMP over TCP reduces the latency by removing the limitation on message size and by allowing several segments of data to be in transit at the same time (due to the TCP window mechanism). TCP has the additional advantage of taking care of retransmission. This may simplify management applications since retransmission need not be implemented at the application level. Both the Linux-based Carnegie Mellon Univ. SNMP library and the UC Davis (UCD) SNMP software have been modified to permit SNMP to run over TCP [5].

The focus of our work is the integration of Transport Layer Security (TLS) into the transport mapping of SNMP over TCP. When SNMP is run over UDP, using TLS for security is not an option; instead IPSec can be used at layer 3. However, if SNMP is implemented over TCP for performance benefits as described above, then TLS is a natural choice. However, it must be determined that the additional overheads associated with TLS/TCP session set up and TLS security are not excessive. This is the motivation for our work in which we implement SNMP over a TLS/TCP base and carry out an experimental analysis of its performance.

Our experiments indicate that:

(1). When the number of messages in one session is large enough (such as 500), the TLS/TCP set up overhead per message is not excessive--approximately 20%.

(2). The TLS security overhead for a session is also not large. This overhead increases when the number of messages in the session increases. When the message number is small, only about 5% of the session time is used in security. For sessions with more than 1000 messages, the security overhead levels off at about 30%, not unacceptable. But the actual overhead per message decreases for longer sessions. Furthermore, using a more powerful computer will reduce the overhead; our experiments were done using a relatively slow workstation (SUN Sparc 10).

(3). SNMPv3/TLS/TCP without User-Based Security Model (USM) is much more efficient than SNMPv3/UDP (with USM) and SNMPv3/TCP (with USM), for similar security features.

2. Background on TLS

The TLS protocol provides communication security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. TLS is based on Secure

Socket Layer version 3 (SSLv3). TLS has been standardized by the Internet Engineering Task Force (IETF).

The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: (1) The TLS Record Protocol. (2) The TLS Handshake Protocol, TLS Change Cipher Specification Protocol and TLS Alert Protocol.

At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the TLS Record Protocol. The TLS Record Protocol provides connection security that has two basic properties:

- The connection is private. Symmetric cryptography is used for data encryption (e.g., DES, 3DES, RC4). The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record Protocol can also be used without encryption.
- The connection is securely reliable. Message transport includes a keyed cryptographic message authentication check (MAC). Secure hash functions (e.g., SHA, MD5) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.

The TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA, DSS). This authentication can be made optional, but is generally required for at least one of the peers.
- The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.
- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

One advantage of TLS is that it is application protocol independent. Higher level protocols can layer on top of the TLS Protocol transparently. TLS runs over TCP/IP.

3. Implementation of SNMP/TLS/TCP

We implemented SNMP/TLS/TCP based on UC Davis UCD-SNMP source code [8]. OPENSSL was used as the TLS (SSLv3.0) source codes [9]. Because the SNMP software of UCD-SNMP can run over TCP, we only need to implement TLS into the SNMP/TCP structure. First we determined the TLS protocol interface in SNMP/TCP structure. Then we implemented the TLS protocol into SNMP/TCP.

The TLS protocols consist of four parts: TLS Handshake Protocol, TLS Change Cipher Spec Protocol, TLS Alert Protocol and TLS Record Protocol.

TLS Handshake Protocol and TLS Record Protocol are implemented over TCP socket. First, a TCP connection is set up. Second, a TLS connection is set up over the TCP connection. Then the client and server begin communication using TLS/TCP. The TLS Change Cipher Spec Protocol causes the pending cipher state to be copied into the current cipher state, which updates the cipher suite to be used on the next connection. This protocol is not necessary in every new connection. The TLS Alert Protocol is used to convey TLS-related alerts to the peer entity. When a TLS-related error occurs, the corresponding alert will be send to the peer. We give the implementing structure of SNMP/TLS/TCP in Figure 1.

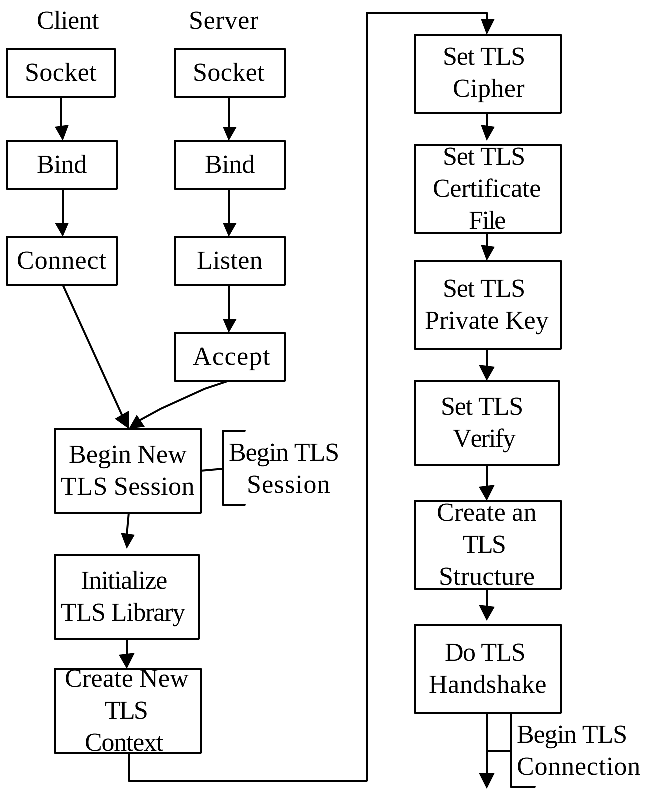

Figure 1: SNMP/TLS/TCP program structure

The TLS handshake protocol and TLS Record Protocol are given in detail as following in Figure 2.

```
=============Begin TLS Handshake Protocol =============
        Client                                    Server
TLS_client_hello                  ---->    TLS_get_client_hello
TLS_get_server_hello              <----    TLS_send_server_hello
TLS_get_server_certificate        <----    TLS_send_server_certificate
TLS_get_key_exchange              <----    TLS_send_server_key_exchange
TLS_get_certificate_request       <----    TLS_send_certificate_request
TLS_get_server_done               <----    TLS_send_server_done
TLS_send_client_certificate       ---->    TLS_get_client_certificate
TLS_send_client_key_exchange      ---->    TLS_get_client_key_exchange
TLS_send_client_verify            ---->    TLS_get_cert_verify
TLS_change_cipher_spec            <-->     TLS_change_cipher_spec
Finished                          <-->     Finished
==============End TLS handshake protocol ==============
                                    |
===============Begin TLS Record Protocol ==============
                        TLS Read OR TLS Write
 (1) Fragment Data/Reassemble Data;  (2) Compress/Decompress;  (3) Calculate
client/server MAC; (4) Encrypt/Decrypt; (5) Append/Remove TLS Record Header.
================End TLS Record Protocol ===============
===================End TLS Session========================
```

Figure 2: TLS Handshake Protocol and TLS Record Protocol

4. Performance Tests and Results

The major performance issues are the overhead of TCP vs UDP, the overhead of TLS and the comparison of SNMPv3/TLS/TCP and SNMPv3/TCP with USM. We ran several experiments to measure these overheads. In our tests, we ran the SNMP Management Station in a SUN Sparc 10 workstation, and the SNMP Agent in a SUN Sparc 5 workstation. The results are clearly platform dependent. For example, the TLS Setup Time is about 300 ms in Sparc 5 while in a Sparc 10 it is about 160 ms.[1]

Measurement Environment :

Network:	Ethernet 10 Mbit.
Hardware:	One Sun Sparc 10 workstation , 128M RAM; One Sun Sparc 5 workstation , 128M RAM.
Software:	Sun Solaris 2.6 OS; SNMP/TLS/TCP and SNMP/UDP software (SNMP can be SNMPv1, SNMPv2c and SNMPv3).

[1] All times are measured in milliseconds.

4.1 Overhead of TLS Security

We compared SNMPv1/TLS/TCP with no security, with integrity protection only, and with both integrity and privacy protection. There are three main security related operations that introduce overhead into TLS: MAC computation, compression, and encryption. There are four corresponding situations to investigate:
(a). No security: no compression, no MAC, no encryption.
(b). Integrity protection only: no compression, has MAC, no encryption.
(c). Privacy protection only: has compression[2], no MAC, has encryption.
(d). Integrity and privacy protection: has compression, has MAC, has encryption.

We use X.509v3 certificates, and the key exchange method is RSA. The length of the key used to sign the certificate is 512 bits. The MAC algorithm we used is MD5 while the encryption algorithm is DES.

We performed tests that used short sessions (single message exchange) as well as tests that used long sessions. For the tests with short sessions, **snmpget** was used to get system.sysName. There is only one message out of and into the management station. For the tests with long sessions, **snmpwalk** (walk SYSTEM object) was used to get all variables in the SYSTEM object. There are 34 messages out of and into the management station. Since each of the four scenarios (a,b,c,d) were performed separately for snmpget and snmpwalk, there were eight experiments in all. Each experiment was run 10 times. The Row 3 (Snmpwalk/msg) in Table 1 denotes the time of row 2 divided by the number of messages (34). The Unix system function **gettimeofday** was used to obtain the session times. The mean latencies are reported in Table 1.

Time	a	b	b - a	c	c-a	d	d - b	d - a
Snmpget	774	805	31	823	49	840	35	66
Snmpwalk	1,044	1,120	76	1,186	142	1,273	153	229
Snmpwalk/msg	31	33	2.2	33	2.1	37	4.4	6.7

Table 1: Session times for short and long sessions

In the short session case (snmpget), compared to the total session time (d), both the latency associated with integrity (b-a) and that associated with privacy protection (d-b) are not large. Integrity protection takes about 4.01% of the session time, and the privacy protection takes about 4.52%. The total security takes 8.53% of the session time.

In the long session case (snmpwalk), Integrity protection takes about 7.28% of the session time, while adding privacy protection takes additional 14.66%. The total security overhead is about 21.94% of the session time. The larger percentage of the total latency taken up by security in the case of the long session can be explained as follows: The setup times for SNMP, TCP and TLS are incurred only

[2] Actually, there is no compression algorithm currently used in TLS and SSL. So none of our experiments include compression.

once per session. But the MAC and encryption overheads are incurred for each message in the session and hence are substantially larger for the long session.

Notice that while the security overhead increases as a portion of the total latency for longer sessions, the actual latency per message decreases for longer sessions.

4.2 Overhead of TLS/TCP Session Setup

We also compared SNMPv1/TLS/TCP without security to SNMPv1/UDP. This is to evaluate the extra costs of TLS and TCP setup time. We did it for two different traffic regimes.

4.2.1 Short sessions (a few seconds) and long sessions (several minutes)

A major issue with SNMP/TLS/TCP is the substantial overhead for setting up a session. When beginning a new TLS session, the TLS handshake protocol is used. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in a TLS record. The TLS handshake protocol produces a significant overhead. In contrast, SNMP/UDP does not incur this penalty.

However, if a management session lasts for a long time, several minutes, or several hours, during this time hundreds or thousands of SNMP messages can be exchanged. Thus, for a long session the costs of setting up the session are amortized over a large number of messages and therefore amount to only a small amount of overhead per message.

We did an experiment that compared the total elapsed time for SNMPv1/TLS/TCP with that for SNMPv1/UDP for sessions ranging from a few seconds (and only a couple of messages) to sessions lasting a few minutes (and exchanging thousands of messages). Session times were measured the same way as in (**4.1**) above.

Since SNMPv1/UDP does not provide any security, a fair comparison of overheads is obtained by using SNMPv1/TLS/TCP with peer entity authentication at session setup time, but without integrity or privacy protection for the SNMP messages. In Table 2, the "TLS mean time per message" is the TLS session time divided by the number of messages in the session. A similar statement applies to the "UDP mean time per message."

We also measured the corresponding times in secure SNMPv1/TLS/TCP case, which means TLS with all the securities. That is to show the overall overheads of TLS, including both TLS/TCP setup overhead and TLS security overhead. We use "Secure TLS" to denote TLS with security.

From the results we can see that the TLS set up time is almost a constant, about 300 ms.

Message # in one session[3]	5	20	50	100	500	1000	1500	2000
Secure TLS session time	929	1,046	1,284	1,542	4,248	7,380	9,871	13,784
Secure TLS setup time	309	315	308	321	294	305	325	296
Secure TLS mean time per message	186	52	26	15	8.7	7.0	6.6	6,9
TLS session time	881	953	1,135	1,409	3,665	6,590	8,779	12,207
TLS setup time	310	307	311	294	313	295	302	292
TLS mean time per message	176	48	23	14	7.3	6.6	5.9	6.1
UDP session time	535	636	780	1,000	2,987	5,456	7,231	10,093
UDP mean time per message	107	32	16	10	6.0	5.5	4.8	5.0
Ratio: Secure TLS time to UDP time	1.738	1.645	1.646	1.542	1.422	1.353	1.365	1.367
Ratio: TLS time to UDP time	1.647	1.499	1.455	1.409	1.227	1.208	1.214	1.210

Table 2: Times in short sessions and long sessions

When the message number in one session is small, the TLS message time is about 1.4 ~1.6 times the UDP message time. As the message number increases, the ratio declines to approximately 1.2 for sessions containing at least 500 messages. Comparing the Secure TLS time with UDP time, it shows that the TLS overheads are not large, especially when the message number is big, the overheads are acceptable.

We can explain these results as follows: Assume there are M messages in one session, and we use P1, P2 to represent processing time per message (send or receive) using TLS and UDP respectively. We can write the TLS session time and UDP session time roughly as:

TLS session time = SNMP setup time + TLS/TCP handshake time + M * P1 + TLS/TCP close time + SNMP close time

UDP session time = SNMP setup time + M * P2 + SNMP close time

In both cases, the SNMP setup time and SNMP close time are the same. Thus,

TLS session time - UDP session time = TLS/TCP handshake time + TLS/TCP close time + M *(P1 - P2)

Let S be TLS/TCP handshake time + TLS/TCP close time, i.e., the TLS setup time. Then the time difference per message between TLS and UDP is given by

[3] To get more than 1 messages in one session, we revised the UCD-SNMP software so that it can repeat send and response messages in one session.

(TLS session time - UDP session time)/M =S/M + (P1 - P2) (1)

Because we use the same message (snmpget public system.sysName), P1-P2 can be considered as a constant. Also as mentioned before, the TLS setup time S is a constant. From (1) we can see that when M increases from 5 to 500, the time difference between TLS and UDP decreases. S is about 300 ms. When M is 1000, S/M is only 0.3 ms. Compared to the total message time 6.6 ms, S/M is only 4.55% which can be neglected. For values of M exceeding 500, the ratio of the per messages times for TLS and UDP becomes essentially constant.

4.2.2 Light traffic (1 message/5 minutes) and heavy traffic(1 message/1 second)

For the light traffic, we generated an SNMP transaction once every 5 minutes, and the light traffic lasted about 60 minutes. We did this for both TLS/TCP and UDP. In the case of TLS/TCP each transaction represented a separate TLS session. The same experiment was repeated in heavy traffic corresponding to an SNMP transaction once every second where a new session was set up for each message.

Note that "light" and "heavy" refers only to the SNMP traffic, not to other network traffic. The results in Table 3 show that the messages require less time in heavy traffic as compared to light traffic. One explanation is that in heavy traffic the SNMP operation runs more often so the related variables in memory can be accessed more quickly. In Table 3 "TLS/TCP" and "UDP" refers to TLS/TCP session time and UDP session time respectively.

Time	TLS/TCP	TLS Setup	UDP
Light traffic (/5mins.)	648	296	123
Heavy traffic (/sec.)	424	273	91

Table 3: Light traffic and heavy traffic

4.3 More Detailed Timing Analysis

We used the UNIX system packet timing tool SNOOP to obtain more detailed timing analyses for TLS/TCP and UDP as illustrated in Figure 5.

Session setup times and the processing times for each packet in an SNMP Walk were determined. The results are shown in Tables 4 and 5 .

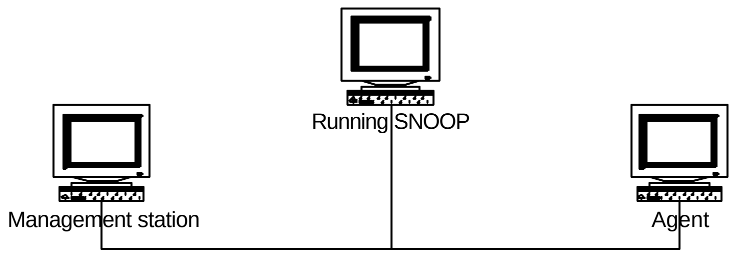

Figure 5: Using three computers to measure the timing

Time	TLS/TCP Setup	SNMP Setup	SNMP Walk	TLS Close	SNMP Close
TLS/TCP	312	521	173	2.4	3.6
UDP		547	112		0.7

Table 4: Time of different session phase

Message	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
TLS	43.8	3.6	3.5	3.4	3.5	3.4	3.5	3.7	3.7	3.5	3.5	3.6	3.6	3.6	3.9	3.6	3.7	3.6	3.6	3.8	3.7	3.7	3.7	3.7	3.9	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6	3.7
UDP	5.4	3.0	3.3	3.1	3.0	3.0	3.1	3.2	3.2	3.1	3.1	3.2	3.2	3.3	3.4	3.9	3.4	3.4	3.2	3.2	3.3	3.3	3.2	3.3	3.3	3.2	3.2	3.2	3.2	3.2	3.3	3.2	3.2	3.4

Table 5: Time of each message

As described earlier, we can divide the time of one SNMP/TLS/TCP session into several parts.

TLS/TCP session time = SNMP setup time + TLS/TCP setup time + several data exchange times + TLS/TCP close time + SNMP close time

Similarly we can divide the time of one SNMP/UDP session into several parts.

UDP session time = SNMP setup time + several data exchange times + SNMP close time

We can see in TLS/TCP case, there is additional TLS/TCP setup time and TLS/TCP close time. The TLS/TCP setup time is nearly constant. Also the individual message exchange times in TLS/TCP are larger than those in the UDP case. The reason for this is that in the TLS/TCP case, the session needs to implement the TLS record protocol when sending and receiving data. That is, the session will do the following operations:

- Fragment Data/Reassemble Data
- Compress/Decompress
- Calculate client/server MAC
- Encrypt/Decrypt
- Append/Remove TLS Record Header.

Also the results show that the first message in both case needs much more time than the later messages. In TLS, the first message took 43.8 ms, about 12.5

times the later message times (3.5 ms). In UDP, the first message took 5.4 ms, slightly larger than the later message times (3.1 ms).

4.4 Comparisons of SNMP/TLS/TCP (without USM) with SNMPv3/TCP and SNMPv3/UDP

SNMPv3 has some security features. It has authentication and encryption. It is interesting to compare SNMPv3(v1)/TLS/TCP (without USM[4]) with SNMPv3/TCP and SNMPv3/UDP when the similar security features are enabled.

SNMPv3 with USM recognizes three levels of security:

- without authentication and without privacy (**noAuthNoPriv**).
- with authentication but without privacy (**authNoPriv**).
- with authentication and with privacy (**authPriv**).

We compared SNMPv3/TLS/TCP with SNMPv3/TCP and SNMPv3/UDP in the above three security levels. The same security algorithms are used in SNMPv3/TLS/TCP and SNMPv3/TCP(or /UDP) to ensure that the comparisons are meaningful. In the tests, MD5 is used as the authentication protocol and DES is used as the encryption algorithm. We also compared SNMPv1/TLS/TCP with SNMPv3/TCP and SNMPv3/UDP in a similar way.

The securityName has a model-independent format, and can be used outside a particular Security Model. Since TLS is based on public key certificates, the securityName in SNMPv3/TLS/TCP is chosen as the identity of the principal in the public key certificate that is used by TLS during the handshaking phase.

As in 4.1, we also performed tests with both short sessions (single message exchange) and long sessions here. For the tests with short sessions, snmpget was used to get system.sysName. There is only one message out of and into the management station. For the tests with long sessions, snmpwalk (walk SYSTEM object) was used to get all variables in the system object. There are 34 messages out of and into the management station. We ran both snmpget and snmpwalk in six cases: SNMPv1/UDP, SNMPv1/TCP, SNMPv1/TLS/TCP, SNMPv3/TLS/TCP, SNMPv3/UDP and SNMPv3/TCP. Also three different security scenarios (a,b,d) were performed separately for the six cases. Table 6.(a) shows the results of snmpget and Table 6.(b) shows the results of snmpwalk. Each experiment was run 10 times. The mean latencies are reported in Table 6. The ratios of SNMPv3/ UDP (or TCP) session times to SNMPv3/TLS/TCP and SNMPv1/TLS/TCP session times are given in the last four rows in Table 6.(a) and Table 6.(b). Also the SNMPv 1/UDP and SNMPv1/TCP session times are given as reference.

Notes: In both Tables 6.(a) and 6.(b), the a ,b ,d in the 1st row have the same meaning as in Table 1. SNMPv1 in case a corresponds to SNMPv3 with **NoAuthNoPriv** security level. SNMPv1 in case b corresponds to SNMPv3 with **AuthNoPriv** security level. And SNMPv1 in case d corresponds to SNMPv3 with **AuthPriv** security level.

[4] In section 4.4, all SNMPv3/TLS/TCP and SNMPv1/TLS/TCP are without USM, while SNMPv3/TCP and SNMPv3/UDP have USM.

SNMP-v1 security feature	a	b	b - a	d	d - b	d - a
Or corresponding SNMP-v3 security level	NoAuth NoPriv	Auth NoPriv		Auth Priv		
Snmpget-v1/UDP	472					
Snmpget-v1/TCP	523					
Snmpget-v1/TLS/TCP	774	805	31	840	35	66
Snmpget-v3/TLS/TCP (no USM)	976	989	13	1,124	135	148
Snmpget-v3/UDP (USM)	665	1,632	967	2,735	1,103	2,070
Snmpget-v3/TCP (USM)	881	1,990	1,109	3,634	1,644	2,753
Ratio :v3-UDP / v1-TLS -TCP	85.9%	203%		326%		
Ratio :v3-UDP / v3-TLS -TCP	68.1%	165%		243%		
Ratio :v3-TCP / v1-TLS -TCP	114%	247%		433%		
Ratio :v3-TCP / v3-TLS -TCP	90.3%	201%		323%		

(a). snmpget

SNMP-v1 security feature	a	b	b - a	d	d - b	d - a
Or corresponding SNMP-v3 security level	NoAuth NoPriv	Auth NoPriv		Auth Priv		
Snmpwalk-v1/UDP	678					
Snmpwalk-v1/TCP	762					
Snmpwalk-v1 TLS/TCP	1,044	1,120	76	1,273	153	229
Snmpwalk-v3/TLS/TCP (no USM)	1,063	1,135	72	1,323	188	260
Snmpwalk-v3/UDP (USM)	648	1,848	1,200	2,976	1,128	2,328
Snmpwalk-v3/TCP (USM)	947	2,025	1,078	3,305	1,280	2,358
Ratio :v3-UDP / v1-TLS -TCP	62.1%	165%		234%		
Ratio :v3-UDP / v3-TLS -TCP	60.9%	163%		225%		
Ratio :v3-TCP / v1-TLS -TCP	90.7%	181%		260%		
Ratio :v3-TCP / v3-TLS -TCP	89.1%	178%		249%		

(b). snmpwalk

Table 6: SNMPv3 (v1)/TLS/TCP vs SNMPv3/TCP and SNMPv3/UDP

Since SNMPv1 has no security, there is only one result in SNMPv1/UDP and SNMPv1/TCP. When security level is NoAuthNoPriv, the v3-UDP session times are always smaller than v3 (v1)-TLS-TCP session times. This can be explained by the TLS and TCP setup times in the later cases. And there is no large difference between the session times of v3-TCP and v3 (v1)-TLS-TCP (ranging from 89.1% to 114%).

But when security is added, the Snmpget-v3/TCP session time is much larger than (from 163% up to 433% of) Snmpget-v1/TLS/TCP session time.

Our experiments show that SNMPv3 (v1)/TLS/TCP without USM are more efficient than SNMPv3 (with USM)/UDP and SNMPv3 (with USM)/TCP, when using similar security features.

5. Conclusion

We have constructed an implementation of SNMP on a TLS/TCP base and conducted experiments to determine whether the additional overhead so introduced is acceptable. Our results indicate that both the session set up overhead and per message security overhead are not excessive. Consequently, SNMP/TLS/TCP appears to be a valid choice for secure network management that takes advantage of the efficiency of TCP.

The comparisons of SNMPv3/TLS/TCP with SNMPv3/TCP (UDP) indicate that both SNMPv3/TLS/TCP and SNMPv1/TLS/TCP are more efficient than SNMPv3 (with USM)/UDP and SNMPv3 (with USM)/TCP, for similar security levels. However, at present it is not clear to what extent this apparent advantage is structural and to what extent it may reflect different degrees of code optimization. Further experience, with different software implementations is needed to verify the generality of the observed results.

Acknowledgment

We would like to thank John Blake (U.S. Department of Defense) and Gary Hayward (Telcordia Technologies) for helpful discussions.

References

[1] J. Schoenwaelder, "SNMP over TCP Transport Mapping" , Internet Draft, draft-irtf-nmrg-snmp-tcp-04.txt, Apr. 2000.
[2] William Stallings, "SNMP, SNMPv2, SNMPv3 and RMON1 and 2", 3rd ed. Addison-Wesley Longman, 1999.
[3] T. Dierks and C. Allen , "The TLS Protocol Version 1.0", RFC 2246, Jan. 1999.
[4] C. Kaufman , R. Perlman and M. Speciner, "Network Security", Prentice Hall PTR, 1995.
[5] R. Sprenkels and J-P. Martin-Flatin, "Bulk Transfers of MIB Data", *The Simple Times,* 7 , Mar.1999.
[6] Michael Boe and Jeffrey Altman, "TLS-based Telnet Security", Internet Draft, draft-ietf-tn3270e-telnet-tls-05.txt, Oct. 2000.
[7] Paul Hoffman, "SMTP Service Extension for Secure SMTP over TLS", Internet Draft, draft-hoffman-rfc2487bis-04.txt, Oct. 2000.
[8] UCDavis UCD-SNMP source code, ucd-snmp 4.1.2, http://net-snmp.sourceforge.net, May 2000.
[9] OPENSSL source code : OpenSSL 0.9.6, http:// www.openssl.org , Sep. 2000.
[10] Eric Young , "SSLeay Documentation", http://www.columbia.edu/~ariel/ssleay/ , Feb.1999.
[11] Ben Laurie , Apache-SSL source code and documentation, http://www.apache-ssl.org/, Nov. 2000.

[12] J. Case, R. Mundy, D. Partain and B. Stewart , "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, Apr. 1999.

[13] U. Blumenthal and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) ", RFC 2274 , Apr.1999.

[14] B. Wijnen, R. Presuhn and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) ", RFC 2575, Apr. 1999.

Xiaojiang Du received his B.S and M.S. degrees from Tsinghua University, Beijing, China in automation in 1996 and 1998 respectively. He is currently pursuing the Ph.D. degree at the University of Maryland, College Park.

During the academic years 1999-2001 Mr. Du is a recipient of a fellowship from the University of Maryland at College Park. His main research interests are in the management of communication networks and network security.

Moshe Rozenblit is a Principal Systems Engineer at TeraBurst Networks in charge of optical network management. Previously he was with the Bell System and its descendents (Bell Laboratories, NYNEX Science and Technology, Bellcore, Telcordia Technologies) for 25 years.

He holds two US patents and has recently published a book - Security for Telecommunications Network Management (IEEE Press). Moshe has BA and MA in physics from the University of Brussels, Belgium, a PhD in physics from Stevens Institute of Technology and an MS in computer science from Rutgers University.

Mark Shayman received his Ph.D. in Applied Mathematics from Harvard University in 1981. Since then, he has been a faculty member at Washington University (St. Louis) and the University of Maryland at College Park, where he is a full professor in the Department of Electrical and Computer Engineering.

Professor Shayman has received the Donald P. Eckman Award from the American Automatic Control Council and Presidential Young Investigator Award from the National Science Foundation. He has served as Associate Editor of IEEE Transactions on Automatic Control. His research interests are in the control and management of communication networks, including performance, fault and security management.