17

Secure Service Management in Virtual Service Networks

Hai Qu
Computer and Information Science Department
University of Delaware, Newark, DE 19716, USA
Tel: +1 619 6511543 Fax: +1 619 6582243
Email: qu@cis.udel.edu

Tuncay Saydam
Computer and Information Science Department
University of Delaware, Newark, DE 19716, USA
Tel: +1 302 8312716 Fax: +1 302 8318458
Email: saydam@cis.udel.edu

Abstract

This paper extends our discussion and treatment of security of service management applications. After a brief discussion of the key players within the secure service management environment, it presents in detail the security service protocol and application programming interface (API) to facilitate request and response between a service management application and its security server. Guidelines on defining service management protocol and extending current CMIP's security management functions are also given. Conclusions encapsulate the results so far achieved as well as the future work and directions of our study.

Keywords

Service Management, Network Management, Security Services, Security Service Protocols, Security Server, ASN.1.S, SP, ACSE_S, CMIP, Security Management, Security MIB.

1 INTRODUCTION

Service Management is an emerging topic on how to make network users to have friendly, flexible and secure control over QOS of their communications. In this area, security of Service Management is still not fully developed, and more study needs to be done on that

A. A. Lazar et al. (eds.), *Integrated Network Management V*

[PRI93] [PRI94]. Our problem to solve is to protect service management communications from various security threats by providing the following security services: Authentication, Data Confidentiality, Data Integrity, Access Control, Security Audit and Security Alarm.

In [Qu96], we designed the extensions of ASN.1, Presentation Layer and ACSE, which were named ASN.1.S, SP and ACSE_S respectively, in order to provide a generic approach to serve the common needs of protected communications among application processes. In this paper, we will try to solve our problem using this generic approach as the basis, by defining Security Service Protocol and API (Application Programming Interface), giving guidelines for designing Service Management Protocols, and specifying the extensions to be done for CMIP's Security Management functions.

2 EXTENSIONS TO ASN.1, PRESENTATION LAYER AND ACSE

Here we will only briefly introduce how the extensions to ASN.1, Presentation Layer and ACSE were done. The details are in [Qu96].

2.1 ASN.1.S: ASN.1 Extended for Security

We extend the current ASN.1 by introducing new data types as follows. The extended ASN.1 is called **ASN.1.S**. The extended encoding rules are called **BER.1.S**, which is BER.1 plus the new encoding rules for these new data types.

ENCRYPTED Type

This new data type is defined as having tag UNIVERSAL 11, which is not used in current ASN.1 standard. The format for this data type is as follows.

```
ENCRYPTED
   { algorithm              OBJECT IDENTIFIER,
     keyType                KeyType,
     toBeEncrypted          ANY }

KeyType ::= ENUMERATION
              { noKey(0),              -- some checksum needs no key
                secretKey(1),          -- the symmetric secret key
                senderPrivateKey(2),   -- the asymmetric private key
                receiverPublicKey(3),  -- the asymmetric public key
                sessionKey(4),         -- the symmetric temporary key
                CAPrivateKey(5)        -- the assymetric private key }
```

algorithm indicates the encryption algorithms to be used for this data type. The encryption key to be used is indicated by *keyType*, which can be the symmetric secret key, sender's asymmetric private key, receiver's public key or a session key dynamically generated during connection establishment. In the definition of this data type, the actual value for *algorithm*

and *keyType* may or may not be fixed. They can be assigned different values each time the application tries to send encrypted data.

The data to be encrypted is *toBeEncrypted* which can be any ASN.1.S type. The Presentation Layer first encodes it according to BER.1.S into a BIT STRING. Then it encrypts this BIT STRING into another BIT STRING. The final data to be transferred in PPDU includes object identifier value for *algorithm*, enumeration value for *keyType* and the latter BIT STRING. The encoding for ENCRYPTED data type is just like the encoding for a SEQUENCE data type, as follows.

```
[UNIVERSAL 11] IMPLICIT SEQUENCE
                        { algorithm      OBJECT IDENTIFIER,
                          keyType        KeyType,
                          encrypted      BIT STRING }
```

CHECKSUMMED Type

This new data type is defined as having tag UNIVERSAL 12, which is not used in current ASN.1 standard. The format for this data type is as follows.

```
CHECKSUMMED
    { algorithm              OBJECT IDENTIFIER,
      keyType                KeyType,
      toBeChecksummed        ANY }
```

Most features are the same as in ENCRYPTED type. The encoding of this data type is like the encoding of the following.

```
[UNIVERSAL 12] IMPLICIT SEQUENCE
                        { algorithm          OBJECT IDENTIFIER,
                          keyType            KeyType,
                          toBeChecksummed    ANY,
                          checksum           BIT STRING }
```

2.2 SP: Presentation Layer Extended for Security

There is no new Presentation Layer service primitives to be introduced. But we need to add more parameters into P-CONNECT primitives as shown in Table 1. (In the tables of this paper, M means Mandatory, C means Conditional, O means Optional and = means the same value.)

Table 1 New Parameters of SP-CONNECT Primitives

	request	indication	response	confirmation
secretKey	O		O	
initiatorPrivateKey	O			
responderPublicKey	O			
responderPrivateKey			O	
initiatorPublicKey			O	
suggestedSecrecyInfoList	O	O		
suggestedIntegrityInfoList	O	O		
resultSecrecyInfoList			O	O(=)
resultIntegrityInfoList			O	O(=)

2.3 ACSE_S: ACSE Extended for Security

ACSE_S supports authentication itself and also supports negotiation of other security parameters by pass-through to SP. In SA-ASSOCIATE.request, we need to add some new parameters to the existing A-ASSOCIATE.request. Also, a new group of service primitives, SA-AUTHENTICATE, are introduced for both parties to authenticate the other party any time during a connection by requesting certificate, password or challenge number.

Table 2 New Parameters of SA-ASSOCIATE Primitives

	request	indication	response	confirmation
initiatorDN	O	O(=)		
recipientDN	O	O(=)		
initiatorAuthenticator	O	O(=)		
responderAuthenticator			O	O(=)
secretKey	O		O	
initiatorPrivateKey	O			
responderPublicKey	O			
responderPrivateKey			O	
initiatorPublicKey			O	
suggestedSecrecyInfoList	O	O		
suggestedIntegrityInfoList	O	O		
resultSecrecyInfoList			O	O(=)
resultIntegrityInfoList			O	O(=)

3 THE PLAYERS AND THE PROTOCOLS IN THE SOLUTION

In Figure 1, we show all the players and protocols used in the solution.

3.1 Service Management Application and Local S_MIB

Service Management Application is a special network communication application that makes service management functions (value-added services) usable to the users of com-

munication services, e.g. bandwidth management, connection management and QOS management in VPN (Virtual Private Network) and UPT (Universal Personal Telecommunication) [PRI94]. The application processes are running at SMS (Service Management System), NMS (Network Management System) or CPN-NMS (Customer Premises Network NMS). They communicate with each other using P1, the Service Management Protocol, which will be defined later.

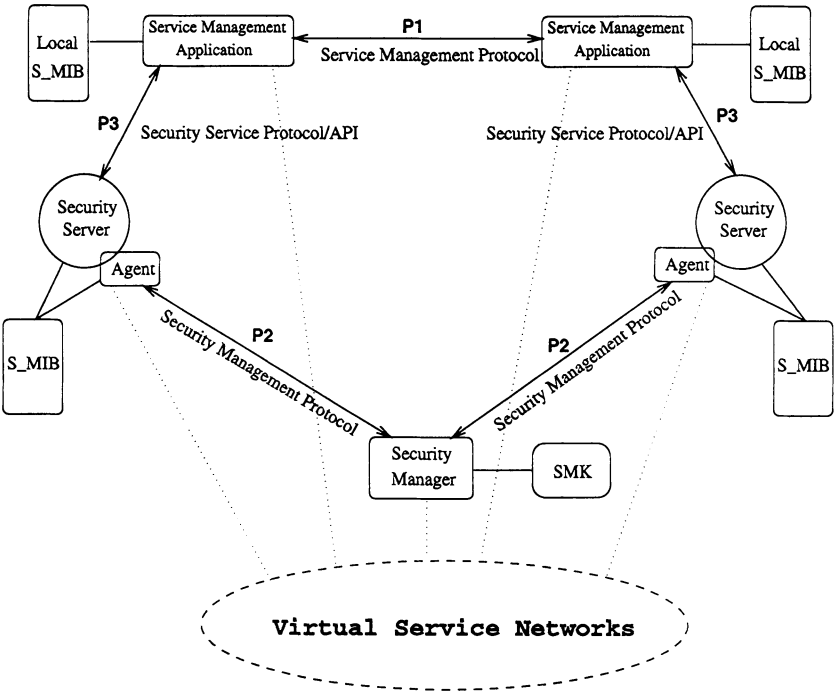

Figure 1 Security of Service Management

Our principal concern here is how to make these application processes to communicate in a secure and controlled way. The Local S_MIB (Local Security MIB) at the Service Management Application will keep some local security information that the application can easily access without contacting the Security Server. The Security Server in the same domain with the application will manage the application's Local S_MIB, using P2, the Security Management Protocol, in a hierarchical management framework, or using some other proprietary management protocol where the Security Server acts as the proxy agent on behalf of the applications which are indirectly managed by the Security Manager.

3.2 Security Server and S_MIB

Security Server logically stores all the security information for its domain in the Security MIB, i.e. S_MIB, and provides security services to Service Management Application upon

request (e.g. providing authentication service) [Muf93]. The application process acquires security service using P3 API which will utilize P3 protocol, the Security Service Protocol, if the Security Server is remote to the application process. If the security server is in the same system as the application process, no open communication protocol activities are involved. A given Security Server can serve more than one application process which are normally in the same domain (e.g. a local area network). S_MIB will contain managed objects such as security policies and domains, security services, security mechanisms, security algorithms, security audit trails, security alarms, etc.

3.3 Security Manager and SMK

Security Manager is the manager role functionality of the security management (i.e. management of security). The corresponding agent role functionality is at the Security Server. The manager and agent communicate using P2, the Security Management Protocol, which is one of the five Specific Management Functional Areas in CMIP [Sta93]. The manager needs a local database SMK (Shared Management Knowledge) that keeps such information as the structure of all S_MIBs it can manage, cached information about the roots of subtrees in S_MIBs, information about domain objects, security policies for management, and so on, which will facilitate the manager to perform management operations over security MIB objects on security servers and the applications.

A manager can manage more than one Security Server. The manager can remotely manage the S_MIB objects that correspond to the security service configuration parameters in the Security Server. The management functions include security object creation/deletion/modification, policy/domain management, authentication and access control management, security audit and alarm management, etc.

3.4 P1: Service Management Protocol

Service Management Protocol is used to carry management information between peer service management entities. There are quite a few studies which explore these service management functionalities [Say95]. Since P1 needs to be defined from the scratch, it would be better to define the PDUs using the new ASN.1.S types, ENCRYPTED and CHECKSUMMED, to make selective field data secrecy and integrity possible [Qu96].

3.5 P2: Security Management Protocol

The following security management functions are already defined among the thirteen Systems Management Functions of CMIP which are within the Security Management Functional Area [Sta93]. They are Security Alarm Reporting, Security Audit Trail and Access Control Management. In addition to the above functions, we still need the following functions to make Security Management in CMIP to be complete: Authentication Management, Confidentiality Management and Integrity Management. The formal definitions of the managed objects for these functions will be studied in the future.

3.6 P3: Security Service Protocol/API

One of the main objectives of this study has been the definition of Security Service Protocol/API (P3) to facilitate the security service request and response between service management applications and Security Servers. No matter whether the security server is local or remote to the service management application process, a set of API functions of a programming language, e.g. C or C++, is to be defined for the application to access security service. The actual entities in the application that will call these API functions are SP, ACSE_S and ACSE_S user. If the security server is remote to the application, the API functions will be responsible for acquiring corresponding services from security server if it cannot provide the security services and/or parameters in its Local S_MIB.

Some Internet documents have already defined a Generic Security Service API (GSSAPI) [RFC1508] [RFC1509]. Its goal is to provide a portable programming interface which is independent of underlying security mechanisms. The user can only have pointers or handles for security data. The user simply calls a sequence of API functions and sends opaque *tokens* to peer user through in-band or out-of-band channel. There are some drawbacks in GSSAPI.

- User data can be signed, can be sealed (integrity plus confidentiality), but cannot be encrypted without integrity checksum.
- The security tokens are opaque to users, i.e. they are only byte strings to users. But the way that user sends data over an application protocol makes it necessary to know the structure of tokens in order to embed the tokens into protocol data. This is especially true for applications defined using ASN.1.
- Selective field confidentiality and integrity can not be achieved. There is no way to indicate parts of data to be protected.

Our purpose of P3 API is different from GSSAPI. P3 API presents to user a concrete method to access security services flexibly. The user still has choices to use different mechanisms but has more direct control over the process of security service. These tangible security services are building blocks for more abstract services. These functions will overcome the drawbacks of GSSAPI.

API Functions in C

Here we specify the C function prototypes for the API functions and also describe the functionalities the application may need to access by explaining what the API functions will accomplish. All the following functions return an integer value where 0 means success and other values mean failure.

(1) *int GetSessionKey (int *length, char **sessionKey);*

Given the requested length of session key to be generated, this function will generate a session key or get a session key from security server and set the actual length of the key when returning. This function will be called by SP to generate a session key during connection establishment time.

(2) *int CryptoRequest (enum Algo algo, char *parm, char *key, int size, char *data, int *resultSize, char **resultData);*

Given the cryptographic algorithm, the parameters, the cryptographic key and the data, this function produce new data to be pointed to by *resultData* by performing cryptographic algorithm on the original data. This function will be called by SP.

(3) *int GetRandomNumber (int *length, char **randomN);*

Given the requested length of random number to be generated, this function will generate a random number or get a random number from security server and set the actual length of the result when returning. This function will be called by ACSE_S in order to generate a random challenge number for authentication purpose. SP will also call this function if it is performing some cryptographic algorithm and is in need of a good source of random numbers that is not available in local system.

(4) *int AuthCheck (char *myDN, char *peerDN, AuthData *authData);*

myDN and *userDN* are the DNs (Distinguished Names) in the X.500 Directory Service format. An application process can claim a specific user identity among the possible more than one identities it may act for. Given the DN that API caller is claiming, peer user's DN and the authentication data that peer user provides, this function checks the authenticity of this peer user. This function is called by ACSE_S user to verify if his peer's password or certificate is correct or not.

(5) *int AccessControl (char *myDN, enum Op op, char *subjectDN, char *targetDN);*

Given the operation requested, the subject, i.e. the requester of an operation, the target on which the requested operation will be performed, this function checks them against the access control rules that may be stored locally or in security server, and rejects or accepts the requested operation. This function will be called by ACSE_S user.

(6) *int SecurityAlarm (int type, int cause, char *detector, char *provider, char *additionalInfo);*

When an entity, either SP, ACSE_S or ACSE_S user, detects a security attack, it can generate a security alarm that the event discriminator will send to security manager or security server who will forward this alarm to security manager. This function will be called by SP, ACSE_S or ACSE_S user.

(7) *int SecurityAuditTrail (int type, int cause, char *additionalInfo);*

This function generates a security audit trail which may be logged by the security server. This function will be called by SP, ACSE_S or ACSE_S user.

(8) *int GetMyAuthData (char *myDN, char *peerDN, Authenticator *auth);*

Before it establishes application association with another user with identity *peerDN*, the

API caller claiming *myDN* as its identity calls this function to get information that is necessary for it to make authentication data which will be sent to peer user during association establishment. This function will be called by ACSE_S user.

(9) *int GetCertificate (char *userDN, Certificate *cert);*

The main purpose of this function is for the sender of data to get the public key of peer indicated by *userDN*, for encrypting data to be sent, or for decrypting signed data. This function will be called by ACSE_S which will pass the public key to SP.

(10) *int GetPrivateKey (char *myDN, int *length, char **privateKey);*

The API caller claiming *myDN* calls this function to get its asymmetric private key. The keys can be acquired by means that is out of the scope of our study, or from Security Server through Security Management exchange. This function will be called by ACSE_S.

(11) *int GetSecretKey (char *myDN, int *length, char **secretKey);*

The API caller claiming *myDN* calls this function to get its symmetric secret key. The keys can be acquired by means that is out of the scope of our study, or from Security Server through Security Management exchange. This function will be called by ACSE_S.

P3 Service Specification

P3 uses ACSE_S directly for association control and authentication services. All other services as defined below will be provided through SP-DATA service primitives. We list the parameters for one group of service primitives as an example, which are basically corresponding to the arguments in API functions described earlier.

SS-CRYPTO	request	indication	response	confirmation
algo	M	M(=)		
parm	O	O(=)		
key	M	M(=)		
data	M	M(=)		
result			M	M(=)
resultData			C	C(=)

P3 Protocol Specification

There is no P3 PDU for association establishment, which will be done by using SA-ASSOCIATION, SA-RELEASE, SA-ABORT and SA-P-ABORT primitives directly. SA-AUTHENTICATE primitives could also be used for extra authentication during the association.

There is a simple correspondence between the service primitives and the PDUs for P3. The

request and indication primitives for a service are mapped to one PDU and the response and confirmation are mapped to another PDU.

Below, we only list the names of the PDUs and how the service primitives are mapped into PDUs.

Table 3 P3 PDUs

	req/ind	resp/conf
SS-SESSION-KEY	SESS-RQ	SESS-RE
SS-CRYPTO	CRYP-RQ	CRYP-RE
SS-RANDOM-NUMBER	RAND-RQ	RAND-RE
SS-AUTH-CHECK	AUTH-RQ	AUTH-RE
SS-ACCESS-CONTROL	AC-RQ	AC-RE
SS-ALARM	ALM-RQ	ALM-RE
SS-AUDIT	AUDIT-RQ	AUDIT-RE
SS-AUTH-DATA	AUDA-RQ	AUDA-RE
SS-CERTIFICATE	CERT-RQ	CERT-RE

The ASN.1.S definitions of these PDUs are to be studied in future. Basically the ASN.1.S definitions are corresponding to the parameters in the service primitives. Here we give the PDU definitions for CRYP-RQ and CRYP-RE as examples. We suppose the data integrity function in SP is either used for each whole PDU, or never used in a connection. Hence, in the ASN.1.S definitions, CHECKSUMMED data type is not needed. The security server and the application will, at connection set-up time, negotiate on whether data integrity will be used or not. Encryption is based on selective field approach, therefore ENCRYPTED data type appears in the definitions.

```
CRYP-RQ ::= [APPLICATION 2] SEQUENCE
                { algo       [0] OBJECT IDENTIFIER,
                  parm       [1] ANY OPTIONAL,
                  secretData [2] ENCRYPTED
                                 { DES,
                                   secretKey,
                                   SEQUENCE
                                     { key    OCTET STRING,
                                       data   OCTET STRING } } }

CRYP-RE ::= [APPLICATION 16] SEQUENCE
                { result     [0] Result,
                  resultData [1] OCTET STRING }

Result ::= ENUMERATION
             { success(0),
               algoNotAvail(1),
               invalidParameter(2),
               invalidAuditType(3),
```

```
invalidUser(4),
noAccessRight(5),
wrongPassword(6),
... }
```

Most likely the keys used for both encryption and checksumming will be the symmetric key, because the symmetric keys are easier to manage for systems that are in the same domain than the public/private key pairs. Also, if the Security Server and the Service Management Application are located physically very closely, these two security services will not be used for better performance so that the application could get security services from its Security Server efficiently.

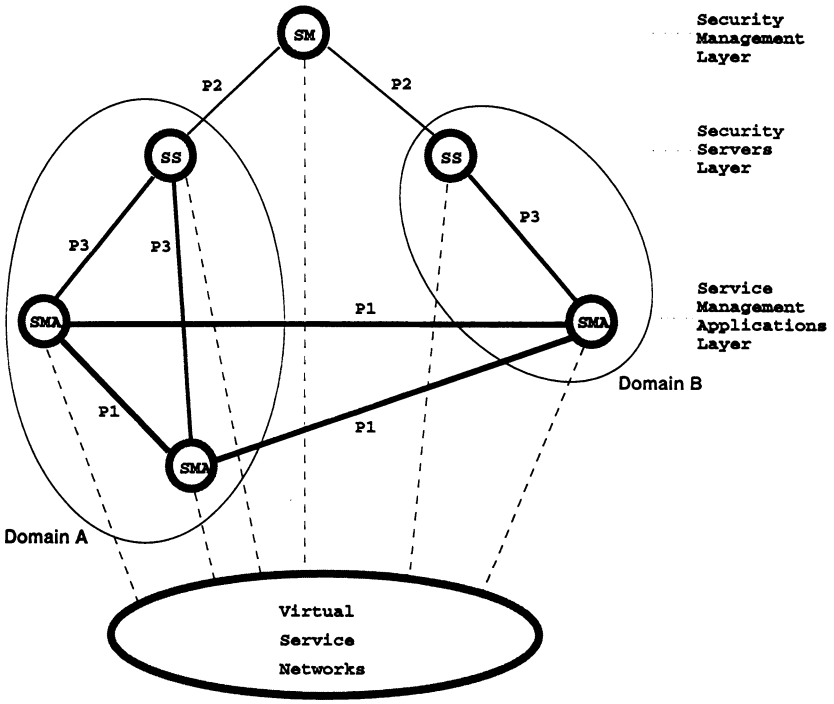

Figure 2 Hierarchical Organization of Security Entities and Domains

4 HIERARCHICAL ORGANIZATION OF THE SOLUTION

As a hint to specifying the solution more formally in the future, here we present the hierarchical organization of entities and domains [Slo94] in our solution as in Figure 2.

SMA, the Service Management Application process, can communicate with other SMAs in

the same security domain or in different domains. The SMAs constitute the Service Management Applications Layer in this logical organization. SS, the Security Server process, will help provide security services to SMAs in the same domain. These SS's constitute the Security Servers Layer. SM, the Security Manager process, which is at the Service Management Layer, will manage SS's directly and SMAs indirectly. The SS's themselves are also organized in another hierarchy, e.g. the hierarchy of certification authorities, which reflects the hierarchical web of trust among the Security Servers. So, domains A and B in Figure 2 may form a bigger domain.

5 CONCLUSIONS

In this paper, we discussed the functions of the players in the solution to the problem of Secure Service Management. Service Management Application processes communicate with each other using Service Management Protocol, which is designed to utilize the security services provided by the underlying SP and ACSE_S. Its security information is managed by its Security Server in the same domain. The Security Server provides security services to SP, ACSE_S and the application process through a newly designed P3 API and P3 protocol which is protected by the underlying SP and ACSE_S. The security information in Security Server is managed by the Security Manager.

Because we utilized the generic security service approach involving ASN.1.S, SP and ACSE_S, there was no duplicate effort done for providing security services to the applications. This unique, comprehensive and flexible underlying approach helps solve our problem very efficiently. Also, P3 API provides a portable programming interface that can eliminate the need for the application to know whether the Security Server is on local system or on a remote system.

Another result we achieved is to have solved the problem of security of OSI network management, i.e. security of CMIP. We don't need to redefine CMIP PDUs and services, but need to implement CMIP entity on top of SP and ACSE_S. Access control function has been partially defined in CMIP; that is, the managed objects for access control have been defined, but how to actually achieve access control has not been defined. We can use Distinguished Names as the identifiers of subjects and targets. Other security management functions to be extended in CMIP were also suggested in this paper.

Our future work will include the perfection of the logical structure of the problem, the complete formal specifications of the three protocols and security-related managed objects definitions in CMIP, and object-oriented modeling [Col94] [Say95] of the solution.

6 REFERENCES

[Col94] Derek Coleman, et al. (1994) *Object-Oriented Development: The fusion method*, Prentice Hall

[Muf93] Sead Muftic, et al. (1993) *Security Architecture for Open Distributed Systems*, John Wiley & Sons Ltd.

[PRI93] *Service Management Reference Configuration*, RACE Project 2041 PRISM, 1993

[PRI94] *VPN and UPT Service Management: Second Case Study Report*, RACE Project 2041 PRISM, 1994

[RFC1508] J. Linn (1993) *Generic Security Service Application Program Interface*, Zolot Associates

[RFC1509] J. Wray (1993) *Generic Security Service API : C-bindings*, DEC

[Qu96] Hai Qu and Tuncay Saydam (1996) *Security of Service Management: A Generic Approach to Secure Communications within OSI*, draft paper to be submitted

[Say95] Tuncay Saydam, et al. (1995) *Object-oriented design of a VPN bandwidth management system*, pp344-355, Integrated Network Management, IV, edited by Adarshpal Sethi, et al., Chapman & Hall

[Sch96] Bruce Schneier (1996) *Applied Cryptography: protocols, algorithms, and source code in C*, John Wiley & Sons, Inc.

[Slo94] Morris Sloman, editor (1994) *Network and Distributed Systems Management*, Addison-Wesley Publishing Company

[Sta93] William Stallings (1993) *SNMP, SNMPv2 and CMIP: The Practical Guide to Network-Management Standards*, Addison-Wesley Publishing Company

7 BIOGRAPHY

Hai Qu is a PhD student in Computer & Information Science Department, University of Delaware. He got B.Sc. and M.Sc. degrees in Computer Science from Fudan University, Shanghai, China. Since his graduate study, he worked on research and development of computer network protocols and applications. He was a Lecturer in Fudan University doing teaching, researching and network application development. He worked in National Computing Center, UK, on OSI protocol testing. He started his PhD study in 1994 working on network management and service management related issues.

Tuncay Saydam is a professor of computer science and computer networks at the University of Delaware since 1979. He has received Dipl.Ing., M.S. and Ph.D. degrees from Istanbul Technical University and the University of Texas. He has been an invited research professor at the Swiss Federale Institute of Technology (ETHZ) and Ecole Polytechnique Federale de Lausanne, where he has been active in research. He has more than 20 years of academic and intensive consulting experience. Author of more than 50 research papers, Prof. Saydam's current research interests include service management, QoS management, virtual networking and telecommunications software engineering.