Non-binary Authentication: Supplicant

HENGCHONG ZHANG

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2009

COS/CCS 2009-01

Non-binary Authentication: Supplicant

Hengchong Zhang

hzh@kth.se

Supervisor and Examiner:
Professor Gerald Q. Maguire Jr.

Master of Science Thesis

Department of Communication Systems
School of Information and Communication Technology
Royal Institute of Technology (KTH)
Stockholm, Sweden

February 24th, 2009

Abstract

There are a number of authentication methods for wireless local area networks. The IEEE 802.1x standard is one such method. This standard specifies a port-based access control protocol. There are three entities involved: a supplicant (a device that wishes to have network access and perhaps other services), an Access Point (AP) or other port to which access is to be controlled, and an Authentication Server (AS). The goal of this project was to design, implement, and evaluate a prototype of a non-binary alternative to IEEE 802.1x authentication. This report focuses on the supplicant. Specifically it describes the design, implementation, and evaluation of a supplicant program to test and stress the authenticator, in order to evaluate a non-binary authentication process.

Following, a brief introduction is given to the problem that is to be solved, a number of existing IEEE 802.1x supplicants are described and compared. Following this, a number of potential non-binary authentication processes are analyzed. The ability of a supplicant to send and receive packets before and after authentication is also examined. Based upon our implementation and evaluation of a supplicant and an emulation of the non-binary authentication process, we conclude that non-binary authentication is both feasible and valuable. Furthermore, the thesis evaluates why and how non-binary authentication is valuable from the viewpoint of a supplicant. Additional future work is suggested at the end of this thesis.

Key words: Supplicant, authentication, IEEE 802.1x, non-binary authentication

Sammanfattning

Det finns ett antal metoder för trådlösa lokala nätverk. IEEE 802.1x-standarden är en sådan metod. Denna standard anger en port-baserad passagekontroll protokollet. Det finns tre enheter som är inblandade: en supplikant (en enhet som vill ha tillträde till nät och kanske andra tjänster), ett Access Point (AP) eller annan hamn som tillgång är att vara kontrollerad, och en Authentication Server (AS). Målet med projektet var att utforma, genomföra och utvärdera en prototyp av en icke-binära alternativ till IEEE 802.1x-autentisering. Denna rapport fokuserar på supplikant. Specifikt beskrivs utformning, genomförande och utvärdering av en supplikant program för att testa och betona authenticator, för att utvärdera ett icke-binära autentiseringsprocessen.

Efter en kort introduktion ges till de problem som ska lösas, ett antal befintliga IEEE 802.1x supplikants beskrivs och jämförs. Efter detta har ett antal potentiella icke-binära autentisering processer analyseras. Möjligheten för en supplikant att skicka och ta emot paket före och efter autentisering är också undersökas. Baserat på vårt genomförande och utvärdering av en supplikant och en emulering av den icke-binära autentisering kan vi dra slutsatsen att icke-binära autentisering är både möjligt och värdefullt. Dessutom, avhandlingen utvärderar varför och hur icke-binära autentisering är värdefull ur ett supplikant. Ytterligare framtida arbetet föreslås i slutet av denna uppsats.

Nyckelord: Supplikant, autentisering, IEEE 802.1x, icke-binära autentisering

Acknowledgement

During the entire thesis project, I received much help, support, and encouragement from several people. Without their help, it would be impossible for me to complete this project. Here, I would like to express my gratitude to all of them.

First, I would like to express my deep and sincere appreciation to my supervisor and examiner: **Professor Gerald Q. Maguire Jr.** He is so knowledgeable, insightful, gentle, and tolerant that I could not have finished this thesis project without his kind help and guidance. In particular I received valuable comments and feedback from him regarding my report. I can't imagine how much time he spent on it. Each time I faced difficulties, he always gave me great assistance with patience. I will never forget his encouraging words which helped me to conquer any challenge. I feel so lucky to have such a good tutor. His positive and easy-going attitude will stay with me all my life.

Second, I would like to thank my colleagues Guo Jia and Zhou Jia. We discussed all of the details of this project and worked together. While each of us worked on different aspects of the problem (with Guo Jia working on the authenticator and Zhou Jia working on the authentication server). Although the time was short, it was an unforgettable experience.

In addition, I am very grateful to my good friends, especially to Guo Yong, Dai Kaiyu, Janko Frick, and Ren Xueliang. Thank you so much for your encouragement and support.

Last but not least, I would like to thank my beloved parents who loved and support me at all times. Any progress or achievement I have made, I would like to share with them.

Table of Contents

List of Figures

List of Tables

List of acronyms and abbreviations

AAA	Authentication, Authorization, and Accounting
ADSL	Asynchronous Digital Subscriber Line
AKA	Authentication and Key Agreement
AP	Access Point
ARP	Address Resolution Protocol
AS	Authentication Server
ASF	Alerting Standards Forum
BSD	Berkeley Software Distribution
CA	Certificate Authority
CCMP	Counter Mode with Cipher Block Chaining Message Authentication Code Protocol
CHAP	Challenge Handshake Authentication Protocol
CSV	Comma Separated Values
DB	DataBase
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EAP	Extensible Authentication Protocol
EAPOL	EAP Over LAN
FDDI	Fiber Distributed Data Interface
GUI	Graphical User Interface
IANA	Internet Assigned Numbers Authority
IAS	Internet Authentication Service
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LEAP	Light EAP
MAC	Medium Access Control
MD5	Message-Digest algorithm 5
NAK	Negative Acknowledge
NAS	Network Access Server
PAE	Port Access Entity
PAP	Password Authentication Protocol
PC	Personal Computer
PDA	Personal Digital Assistant
PEAP	Protected EAP
PPP	Point to Point Protocol
RADIUS	Remote Authentication Dial In User Service
SIM	Subscriber Identity Module

TKIP	Temporal Key Integrity Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
VoIP	Voice over IP

1. Introduction

1.1 Problem Statement

With the rapid development of IEEE 802.11 wireless local area networks (WLANs), both companies and ordinary people have benefited due to the use of wireless data communications technology. Companies reduce the cost of installing wiring and provide greater convenience to their employees. For most users, WLANs enable them to enjoy Internet services easily with a laptop or personal digital assistant (PDA) in a comfortable surrounding.

When a user wants to access an IEEE 802.11 wireless network, a common security feature is to require authentication of the wireless device before providing this user's device with any service[1]. Currently, there are a number of authentication methods for WLANs. In this project, we focus on IEEE 802.1x standard. In Chapter 2, we will examine this standard in detail. Here, we will only give a brief introduction to the authentication process when using IEEE 802.1x in the context of authentication, authorization, and accounting (AAA) for WLANs.

IEEE 802.1x is a port-based access control protocol which was originally designed for the point-to-point protocol (PPP)[2], then adapted to wired network ports and later to WLANs. There are three entities involved: a supplicant, an access point (AP) or other port to which access is to be controlled, and an authentication server (AS). In this report we will focus on infrastructure mode WLANs, hence the resource being controlled is traffic passing through an AP. However, a similar approach can be used to control access via a wired network port (for example, an Ethernet switch). The process begins when a supplicant wants access to the network. After the IEEE 802.11 wireless interface of the supplicant associates with an IEEE 802.11 AP, this AP will only permit the supplicant to access the network after this supplicant is authenticated by the AS and only if it is authorized to access the network. The AS decides whether the supplicant represents a valid user who should get access to the network and in many cases can even decided what sort of access should be granted (for example, how much bandwidth this user's device should receive). We refer to this authentication process as a binary decision, because the AP decides whether to forward or not forward **any** traffic (*other* than authentication traffic) for this supplicant (thus it is a binary decision). Hence, if the supplicant is authenticated by the AS, then the AP will forward traffic otherwise it will not. However, because the time to make this decision is relatively long (in comparison to the inter-arrival spacing of voice over IP (VoIP) packets) a mobile user who moves into the coverage area of this access point will not be able to send or receive packets for some time – in the case of VoIP this will result in a gap in the conversation or if the delay is too longer perhaps even premature

termination of the conversation. Thus the problem with the existing authentication and authorization procedure is that the AP will not provide any service until the device has been both authenticated and authorized, despite the fact that we might expect that most devices will in fact be successfully authenticated -- therefore providing poor services to legitimate users!

1.2 Goals of this thesis

Based on the problem stated above, in this thesis project, we try to design a *non-binary* authentication system as an alternative to IEEE 802.1x. With this system, a supplicant can maintain a connection via the network at a low level (of traffic) – even **without** being authenticated. Thus the user who moves into the coverage of a given AP will be able to send and receive a limited number of packets for a limited period (hence continuing their voice over IP session or other activity) *in parallel* with the authentication and authorization process; therefore, unlike the situation today, the user will not experience any handover latency due to AAA. The proposed authentication system will dynamically manage and control the maximum bandwidth made available to the supplicant, thus improving the perceived performance for valid users, while minimizing the resources given to invalid users.

This thesis project mainly focuses on the supplicant. In particular this thesis was to focus on a supplicant that would like to utilize resources *without* being a valid user, i.e., it tries to sneak more packets through the authenticator than it should normally able to send. The goal of this supplicant is to probe the authentication system to see if it allows the supplicant to send any extra traffic, how much traffic this supplicant can send before it is stopped, and how the authentication system responds when the supplicant sends more traffic than it is expected to send.

This thesis project was done with cooperation with another two thesis projects by Zhou Jia and Guo Jia who are developing other parts of the non-binary authentication system. My specific task was to test and stress Guo Jia’s authenticator.

2. Background and related work

2.1 WPA

The Institute of Electrical and Electronics Engineers (IEEE) standardization working group[3] which focuses on Local Area and Metropolitan Area Networks (i.e., the 802 working group)[4] is further divided into different working groups. One of them is 802.11 which establishes standards for WLAN. At first, IEEE 802.11 defined only one security mechanism: Wired Equivalent Privacy (WEP). Unfortunately, security experts found that there were some major weaknesses in WEP[2,5]. For example, it is vulnerable to malicious tampering of messages and lacks protection against replay attacks. Therefore, a solution was needed to address these problems.

In November 2002, the Wi-Fi Alliance introduced a new security mechanism based on a draft of IEEE 802.11i, in order to quickly promulgate a subset of the proposed security enhancements. This step on the way to IEEE 802.11i is called Wi-Fi Protected Access (WPA)[6,7]. The Wi-Fi Alliance provides interoperability testing and certification of IEEE 802.11 WLAN equipment. Today new WLAN products support WPA. For some older equipment, vendors have developed software upgrades to support WPA - without requiring a change of equipment.

WPA uses the Temporal Key Integrity Protocol (TKIP) which is a more secure encryption method. Also, WPA supports a number of different network modes. In home mode (without an authentication server), users can get access by entering a network key. In enterprise mode, WPA needs an authentication server and it supports IEEE 802.1x and the Extensible Authentication Protocol (EAP). In this thesis we will focus on the enterprise mode - as it is in this mode that the effect of the time required to authenticate a user's device is most significant. In addition, the enterprise mode is most likely going to be used in an environment with large numbers of APs and user devices; hence, the total traffic load to perform authentication and authorization is likely to be significant -- further increasing the delay in performing both processes.

2.2 Authentication for Dial-up access

Before we describe the concepts underlying IEEE 802.1x, it is very useful to introduce an authentication method developed earlier for use with dial-up access users. This authentication method is useful because both EAP and Remote Authentication Dial-In User Service (RADIUS) protocols are based on the concepts developed to support dial-up access.

Although broadband network access is very common nowadays, many users still use dial-up methods to access the internet. They receive a phone number, username, and password from their Internet Service Provider (ISP), and after a simple configuration of their computers (and in some cases their modem) their modem can dial this number and their computer can use this username and password to access the network via this ISP's access network. The connection is a point to point connection between two modems and is based on a protocol called the Point-to-Point Protocol (PPP)[8]. As a data link layer protocol, PPP is used for connections over both synchronous and asynchronous circuits[9]. There are two authentication methods in PPP which are commonly used by ISPs. One method is called the Password Authentication Protocol (PAP). In this method a user's name and password are sent in plain text. This means that eavesdroppers can intercept them easily. The other method is called the Challenge Handshake Authentication Protocol (CHAP). It is based on a challenge response mechanism and offers better security than PAP, but it is still not very secure. To solve this problem, the Extensible Authentication Protocol (EAP) was developed by the Internet Engineering Task Force **(**IETF) and defined in RFC 2284 (PPP Extensible Authentication Protocol)[10]. The details of EAP will be presented in section 2.4.

2.3 IEEE 802.1x

What is IEEE 802.1x? According to the 802.1X-2004 standard, it is described as follows:

"Port-based network access control makes use of the physical access characteristics of IEEE 802 LAN infrastructures in order to provide a means of authenticating and authorizing devices attached to a LAN port that has point-to-point connection characteristics, and of preventing access to that port in cases in which the authentication and authorization process fails. A port in this context is a single point of attachment to the LAN infrastructure." [18]

Therefore, IEEE 802.1x provides compatible authentication and authorization mechanisms for devices interconnected by IEEE 802 LANs and WLANs. Normally, it involves three entities[18]:

Supplicant	The supplicant is an entity at one end of a point-to-point LAN segment that seeks to be authenticated by an Authenticator attached to the other end of that link. Thus the supplicant is an entity which wants to connect via this network. A number of other names are frequently used to describe this entity, such as "user", "client" or "authenticating peer"[2]. In this thesis, we use the term "supplicant" to describe this entity

Authenticator — The authenticator is an entity at the other end of a point-to-point LAN segment that facilitates authentication of the entity attached to the other end of that link. Thus the authenticator controls network access by supplicants.

Authentication Server — An authentication server provides an authentication service to an Authenticator. This service determines, from the credentials provided by the Supplicant to the authentication server, whether the Supplicant is authorized to access the services provided by the system in which the Authenticator resides. Thus it is the authentication server that makes authorization decisions, while the authenticator is in charge of enforcing this decision.

Although our main concern is to apply 802.1x to WLANs, from the definition of 802.1x, we see that 802.1x was originally developed for wired LANs and dialup connections using PPP. In the former case it was used to prevent someone from getting access to the internet simply by plugging a cable into a jack on the wall unless they have authorization. The relationship among the three entities in a dialup PPP scenario is shown in Figure 2.1.

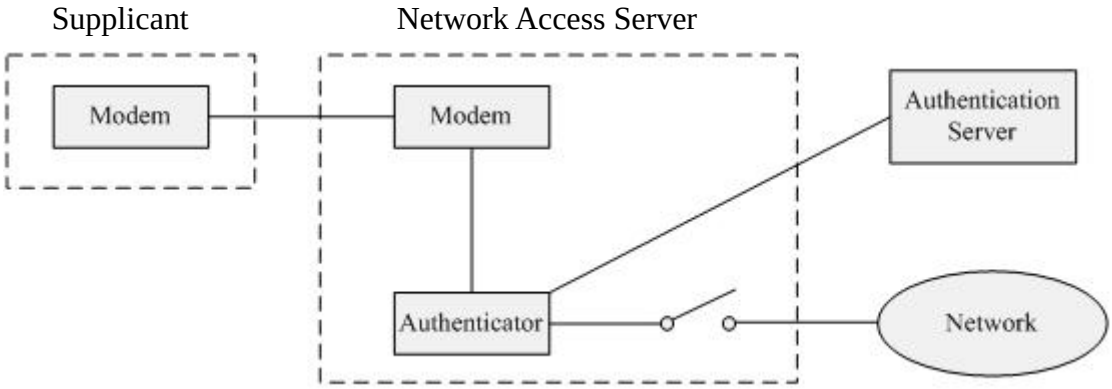


Figure 2.1: Three entities in Wired LANs

As shown in Figure 2.1, the supplicant wishes to access the network by connecting via a Network Access Server (NAS). We will assume that initially, the supplicant does not have access to the network. Therefore, the NAS initially prevents the supplicant from accessing the network. The NAS has no authority to make decisions about whether to let the supplicant access the network or not, instead this decision is made based upon entries in an authentication database operated by the authentication server. Therefore, the authenticator function within the NAS has to communicate with the authentication server before it can change the state of the switch to closed (i.e., granting the supplicant access to the network). Only when the

supplicant has successfully been authenticated by the authentication server, does the NAS close the switch in order to permit the supplicant to access the network.

When applying IEEE 802.1x to wireless LANs (WLANs), the Access Point (AP) takes the place of the NAS and wireless links take the place of physical (cable) connections. The wireless communication between a supplicant and an AP is assumed to be based on the Extensible Authentication Protocol (EAP) or more precisely EAP over LAN (EAPOL). The details of EAP and EAPOL[17], we will be presented in section 2.4. The relationship among the three entities in the case of a WLAN are shown in Figure 2.2.

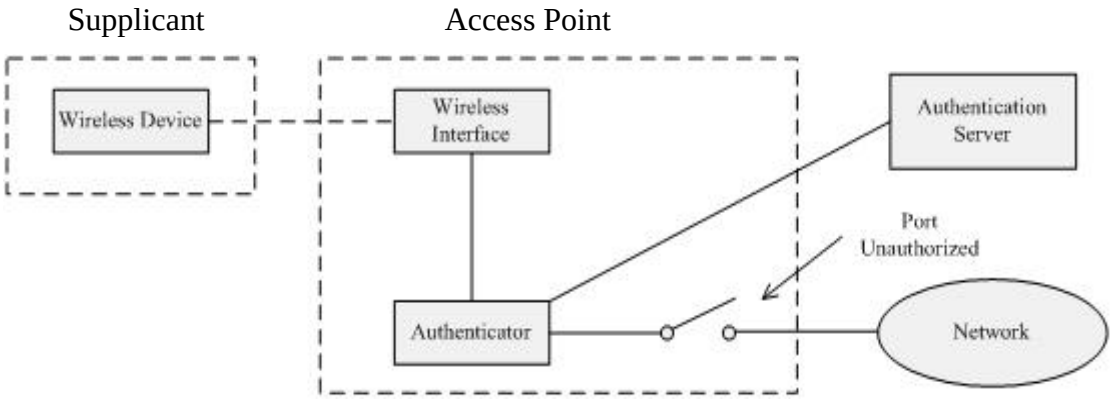


Figure 2.2 Three entities in Wireless LANs

In Figure 2.2, the wireless device, such as a PDA or laptop, is a supplicant which wants to access the network. In order to do so it must respond to the authenticator's request for data. Note that the authenticator is not the AP itself, but could be a component of the AP. In Figure 2.2, the authenticator is shown inside the AP, but it could be also implemented by another device. As in the wired LAN case, once the authentication server authenticates the supplicant, then the authenticator will enable the supplicant to access the network and all packets from the authenticated client may pass through the (logical port) switch.

As stated earlier IEEE 802.1x is a **port-based** authentication method. The authenticator operates controlled and uncontrolled ports. Both the controlled and uncontrolled ports are virtual ports. However, they can share the same physical connection to the LAN simply by filtering out frames which are destined for the network when the logical switch is open[11]. Additionally, both the supplicant and the authenticator have a Port Access Entity (PAE). Before the supplicant is authenticated, the authenticator's controlled port maintains the switch in an unauthorized state. In this state the only traffic allowed to pass is EAPOL. After the supplicant is authenticated, the authenticator's port will be set to the authorized state.

Based on EAP, IEEE 802.1x provides several authentication mechanisms for

WLANs. Examples of these authentication mechanisms are[12]:

- EAP-Message Digest 5 (EAP-MD5)[14].
- Certificate-based solutions such as EAP-Transport Layer Security (EAP-TLS)[13].
- EAP-Tunneled TLS (EAP-TTLS)[15].
- Lightweight Extensible Authentication Protocol (LEAP)
- Smart-card-based solutions such as EAP-Subscriber Identification Module (EAP-SIM).
- Password-based solutions such as EAP-One Time Password (EAP-OTP).
- Protected Extensible Authentication Protocol (PEAP)[16].

2.4 EAP

2.4.1 What is EAP

In previous sections, we have mentioned EAP many times. In this section, we will specify EAP in greater depth following a general introduction.

The Extensible Authentication Protocol (EAP) is defined by RFC 3748 as: “An authentication framework which supports multiple authentication methods. EAP typically runs directly over data link layers such as Point-to-Point Protocol (PPP) or IEEE 802, without requiring IP. EAP provides its own support for duplicate elimination and retransmission, but is reliant on lower layer ordering guarantees. Fragmentation is not supported within EAP itself; however, individual EAP methods may support this.” [17]. The 802.1x protocol stack is shown in Figure 2.3.

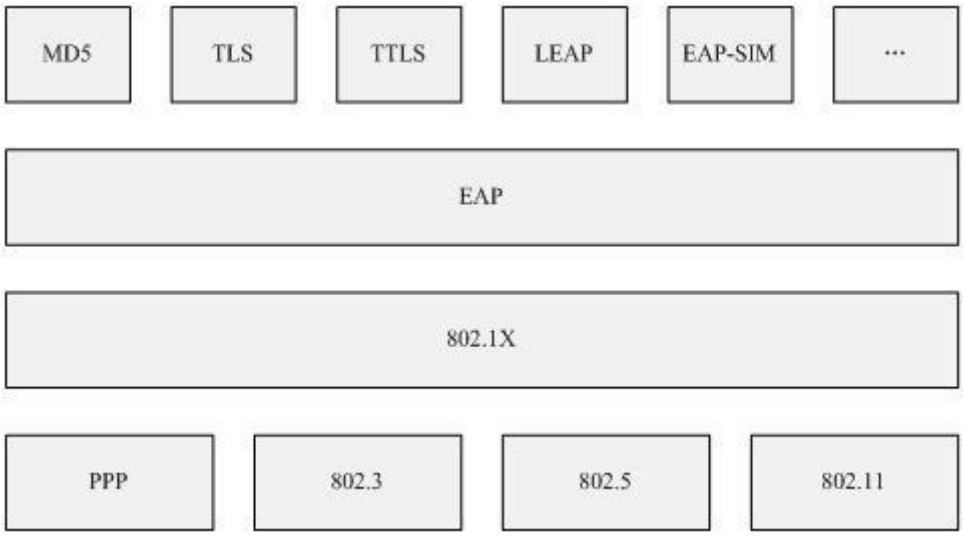


Figure 2.3 802.1x in the context of the surrounding protocol stack

From this definition, we learn that IEEE 802.1x is **not** an authentication method, but rather that it makes use of EAP in order to support several authentication methods: EAP-TLS, EAP-MD5, and so on. Besides these authentication methods, other special methods are also allowed between the supplicant and the authenticator. These methods have not been defined by the standard, but they can be added later if they become popular. That is why EAP is called extensible.

Different types of messages are sent during the authentication process. In RFC 3748, four types of messages are mentioned[17]:

Request	Used to send messages from the authenticator to the supplicant.
Response	Used to send messages from the supplicant to the authenticator.
Success	Sent by the authenticator to indicate access is permitted.
Failure	Sent by the authenticator to indicate access is declined.

All EAP messages have a similar format, shown in Figure 2.4:

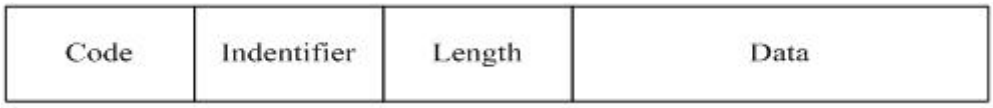


Figure 2.4 EAP message format

The code field is eight bits long and indicates the **type** of the message. Codes 1 to 4 stand for Request, Response, Success, and Failure respectively. The identifier field is an eight bit long value used to match the response with the request. Length is a sixteen bit field indicating the overall number of bytes in the EAP message. The data field is zero or more bytes in length. The contents of the data field are the actual data sent by supplicants and authenticators.

The messages of Success and Failure indicate the result of the authentication server's decision. The Request and Response messages indicate what type of authentication is used. To indicate this, an extra eight bit field called **Type** is added to the basic message format. The resulting format of a Request or Response message is shown in Figure 2.5.

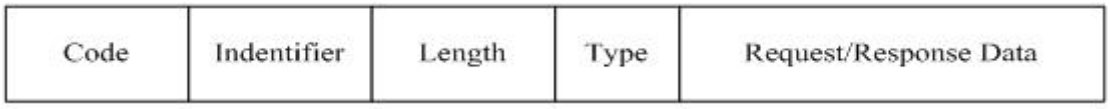


Figure 2.5 Request/Response message format

The type field indicates what information is being carried in the EAP message. Also, it identifies different authentication methods. In RFC 3748, only type values 1 to 6 and 254 to 255 are defined. The other values are available to use for new authentication methods. Type value 1 "Identity" is the basic, but very important

authentication method. Initially, a request message with type value 1 is sent by the authenticator to a new supplicant. The supplicant replies with a response message with type value 1. This response message includes the suppliant's identity information such as a username which will be presented as the supplicant's identity to the authentication server. There are also some special rules for different type values[17]. For example, all EAP implementations must support types 1 to 4 (respectively Identity, Notification, NAK, MD5-Challenge) and type 254 (Expanded types); NAK can only be applied to Response packets. Also, type Identity, Notification, and NAK are considered special case types.

2.4.2 EAPOL

RFC 3748 does not specify how messages are sent among the entities. EAP was originally designed for a dial-up authentication process rather than authentication on a LAN (or WLAN). To define how EAP messages should be transferred on a LAN, a protocol named EAP encapsulation over LANs (EAPOL) is defined in IEEE 802.1x to transport these messages among the different entities.

EAPOL encapsulations have been specified for IEEE 802.3/Ethernet and Token Ring/FDDI. The EAPOL encapsulation used for IEEE 802.3/Ethernet can also be used for other LAN media access and control (MAC) protocols technologies which have the same basic frame format as Ethernet[18]. The frame format of an EAPOL used by Ethernet is shown in Figure 2.6.

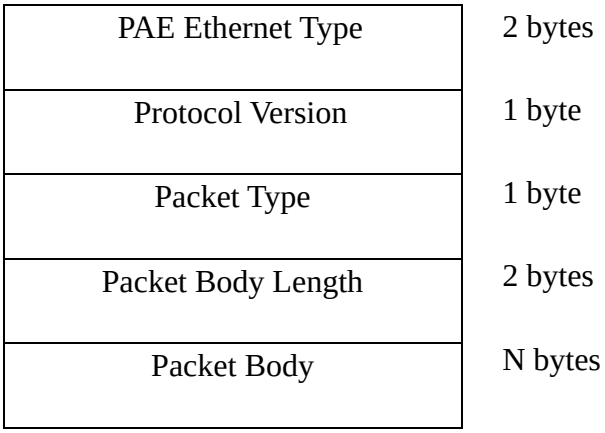


Figure 2.6 EAPOL frame format for Ethernet

There are five types of packets. They are briefly introduced as follows:

EAPOL-Start — If a supplicant wants to access the internet, the first step it needs to take is to find a viable authenticator. However, the supplicant does not know where it can find an available

	authenticator nor does it know the authenticator's MAC address. IEEE 802.1x defined a EAPOL-Start message to solve this problem. The supplicant sends an EAPOL-Start message to a special group-multicast MAC address (01:80:c2:00:00:03) which specifies all authenticators. In this way, the supplicant may find an existing authenticator and notify this authenticator that it is ready.
EAPOL-Logoff	An EAPOL-Logoff message indicates that the supplicant wishes to be cut off from the network. When the authenticator receives this message, it will return the port to the uncontrolled state.
EAPOL-Packet	This type of message is used for sending the actual EAP messages (including Request, Response, Success, and Failure).
EAPOL-Key	This type of message is used for sending session key information between the authenticator and the supplicant.
EAPOL-Encapsulated-ASF-Alert	This type of message is used for allowing alerts to be forwarded through a port which is in the unauthorized state[18]. Currently, it is used by the Alerting Standards Format (ASF) [40].

These messages are sent via EAPOL among supplicant, authenticator, and authentication server. Next, we will examine the EAP message flow and the complete process.

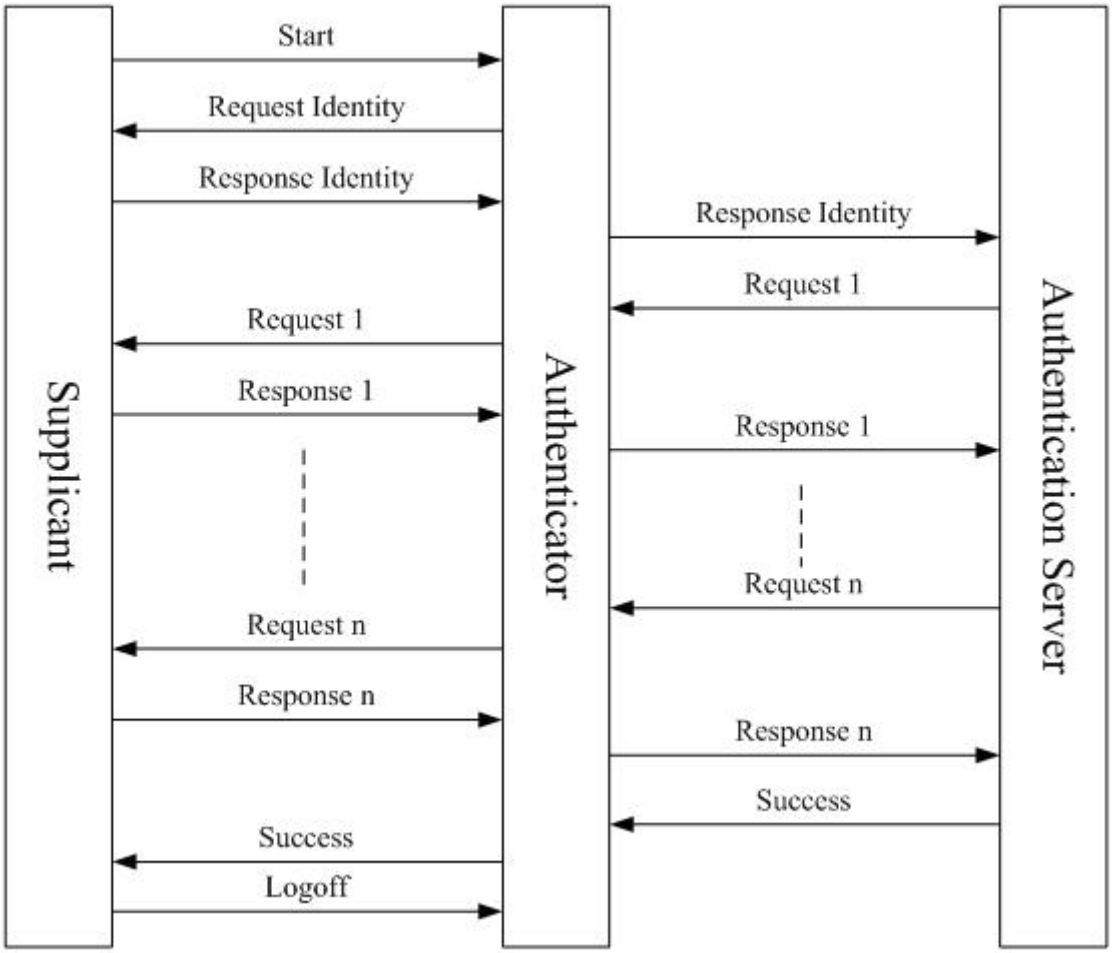


Figure 2.7 EAP message flow

As figure 2.7 shows, if the supplicant wants to access the network, it first needs to find an authenticator. The supplicant sets up a connection with the authenticator by sending an EAPOL-Start message. Once the authenticator is alerted by the supplicant, it will send a Request Identity message back to the supplicant. In some cases, the authenticator does not need to do this if it knows the supplicant's identity by some other methods. Then, the supplicant needs to send a Response Identity message back to indicate its identity. During this process, the authentication server is not (yet) involved. Now that the authenticator knows the identity of the supplicant, it needs to communicate with the authentication server to learn if the supplicant is to be permitted access to the network. As noted earlier the authentication server is responsible for making this decision. Therefore, the authenticator does not need to know which kind of authentication method is used. Its task is simply to forward a Request message from the authentication server to the supplicant and a Response message from the supplicant to the authentication server. The authenticator does not care about the contents of these messages, except for the Success and Failure message from the authentication server. If the authenticator receives a Success message, it will enable the supplicant to access the network, otherwise the supplicant will be denied access.

2.4.3 PEAP

As described in last section, EAP is used for authentication of IEEE 802.1x and it supports several authentication methods. Besides the methods we mentioned earlier, EAP-Double-TLS, EAP-SAKE, and EAP-POTP methods have been proposed in recent years[41]. However, we will not specify all of them as their details are outside the scope of this thesis. Readers who are interested in these methods are referred to [41].

There are also some drawbacks in EAP. For example, the EAP negotiation lacks protection, thus the EAP Identity message can be eavesdropped by an attacker. Also, there is no standardized mechanism for key exchange, no built-in support for fragmentation and reassembly, and no support for fast reconnection[19]. One method to solve these problems is to do the EAP negotiation in a private encrypted way. Thus, the authentication process is much safer and the supplicant's identity will not be disclosed. This is the general principle of PEAP.

Protected Extensible Authentication Protocol (PEAP) is a method to safely send authentication information over wired or wireless networks[19,20]. It was developed by Cisco Systems, Microsoft, and RSA Security as an open standard. Today it is widely used. To make the authentication process more secure, PEAP uses server-side only public key certificates to create a secure TLS tunnel between the supplicant and the authentication server. As a result, the whole authentication process can de divided into two phases: authentication and secure message transport.

In the first phase, the authentication process is similar to the usual EAP negotiation. The authenticator sends a Request packet to the supplicant to inquire its identity. The supplicant replies with a Response packet to the authenticator to state its identity. This identity information indicates which authentication method is to be used. Given this information the authentication server can respond appropriately. However, the supplicant does not actually have to send its real identity in the first phase. Instead, this identity can be transferred in the second phase. During the first phase, TLS is used to setup a safe and private connection between the supplicant and the authentication server. Note that only the server is authenticated in this phase. This is because the server needs to prove its identity in order to be trusted by the supplicants (in order to avoid passing the supplicant identity and credentials to a fake authentication server).

In the second phase, all the EAP messages are sent via the encrypted TLS session established in the first phase. Now the supplicant can reveal its real identity knowing that only the authentication server can see this information. However, the supplicant's identity may not be trusted at the beginning of the second phase since any attacker could perform the TLS negotiation with the authentication server and setup a TLS

connection. Therefore, the supplicant must authenticate itself during the second phase. Note that in addition to authenticating itself for the TLS session the supplicant also authenticates itself to the authentication server for the purposes of gaining access to the network (which was the whole point of the overall process!).

2.5 RADIUS

Remote Authentication Dial In User Service (RADIUS) is a very commonly used component in the authentication process. RADIUS is an authentication, authorization, and accounting (AAA) protocol used for IP networks[21,22]. It was first introduced by Merit network in 1991 in order to control dial-up access. The first RADIUS RFC was published in 1997[44]. This was replaced later by RFC 2865[21]. RADIUS has been widely applied for network access authentication and many open source RADIUS servers are available.

As we mentioned above, RADIUS was initially used for dial-up access control which is based on a point-to-point protocol. PAP and CHAP are the two authentication methods most strongly associated with PPP. When using these two authentication methods, at most four types of packets are sent between the Network Access Server (NAS) and the RADIUS Authentication Server (AS)[21]:

- Access-Request (from the NAS to the AS)
- Access-Challenge (from the AS to the NAS)
- Access-Accept (from the AS to the NAS)
- Access-Reject (from the AS to the NAS)

The PAP authentication method is comparatively simple. First, the dial-up user connects to the NAS. Then the user sends a user name and password to the NAS to prove its identity. The NAS will send an Access-Request message to the RADIUS authentication server which contains the user’s account information. The RADIUS authentication server indicates its decision by sending an Access-Accept or Access-Reject message back to the NAS. This authentication flow is shown in Figure 2.8. However, this method is rather insecure since the password is sent in plaintext and it can be easily captured by malicious persons.

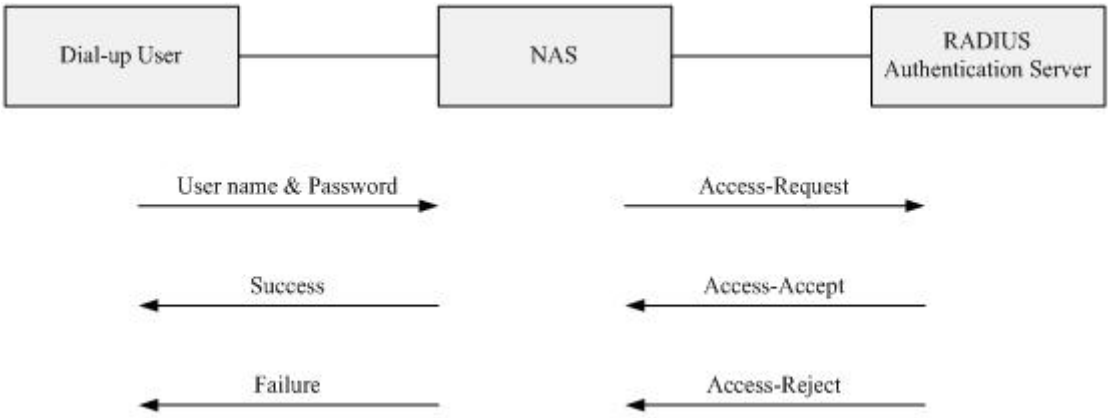


Figure 2.8 PAP Authentication Flow

The other authentication method, CHAP, is a little bit more secure. In this method, the user first sends a user name to the NAS. Then the NAS sends back a challenge to the user. This challenge is a random number. When the user receives the challenge, it encrypts the challenge with its own password to generate a response. The user sends a new response to the NAS. The NAS sends an Access-Request message to the AS by forwarding the challenge, response, and identity information to the RADIUS authentication server and waits for its decision (in the form of an Access-Accept or Access-Reject message). This authentication flow is shown in Figure 2.9.

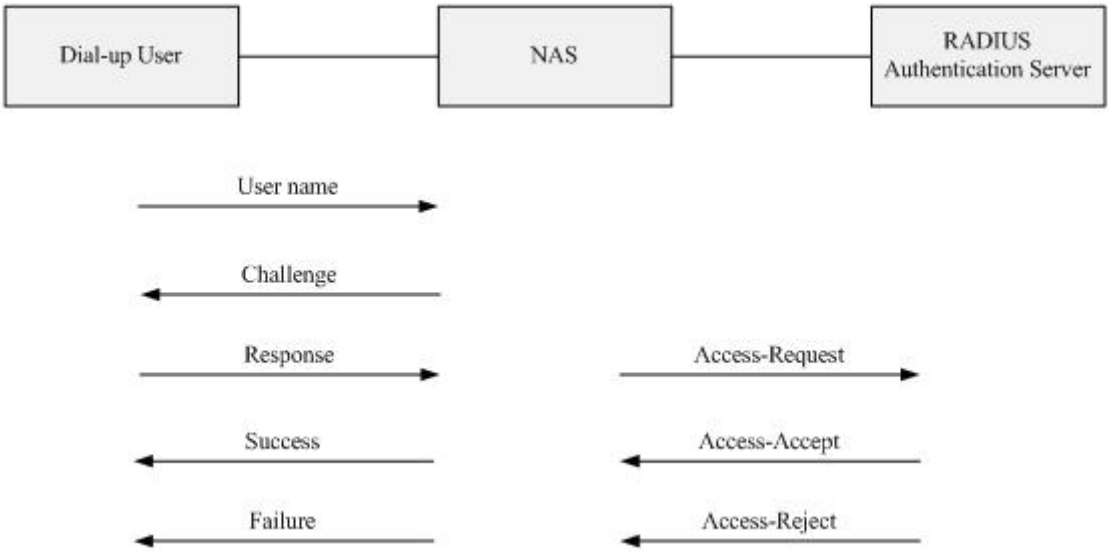


Figure 2.9 CHAP Authentication Flow

The basic format of a RADIUS message[21] is shown in Figure 2.10. The 8 bit code field indicates the type of the message. Code values from 1 to 3 represents Access-Request, Access-Accept, and Access-Reject message correspondingly. While value 11 indicates an Access-Challenge. The 8 bit identifier field is used for matching requests and replies. The 16 bit length field indicates the total number of bytes in the message. The 128 bit authenticator field is used for authenticating the feedback from

the RADIUS server. Each message can carry one or more attributes and each one is a self-contained package of information. In addition, new attribute values can be defined which makes RADIUS extensible. Attribute value 26 enables vendors to implement their own proprietary hardware and software extensions. Using this feature, Microsoft has designed and implemented MS-CHAP to support Microsoft's proprietary dial-up protocols[23].

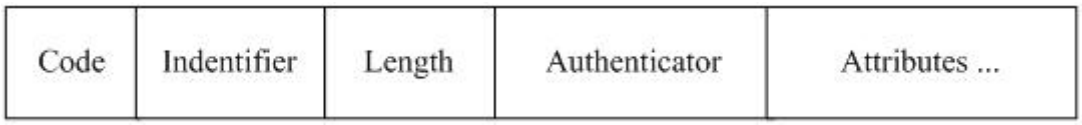


Figure 2.10 RADIUS Message Format

Since RADIUS is an AAA protocol, it also supports accounting. The NAS can make use of this to transfer the following parameters to the authentication server[22]:

- The user's session start time
- The user's session end time
- Reason for the session ending
- Total number of packets transferred during the session
- Volume of data (i.e., number of bytes) transferred during the session

2.6 Related Research

2.6.1 Who is the Supplicant

The term supplicant is a basic concept in IEEE 802.1x. Generally speaking, a supplicant is an entity that is being authenticated by an authenticator. The supplicant can connect to the authenticator through a point-to-point protocol, IEEE 802.3, or IEEE 802.11 link. In practice, supplicants are client software developed by operating system or third-party vendors and installed on end-users' computers. For example, Microsoft provides a supplicant in Windows XP, 2000, ME, and even some earlier releases[24]. There are many 802.1x supplicants, some of which are commercial, while others are available for free. Of the later, some are even open source products, hence the source code is public. In particular, several networking and security technology companies have cooperated as the OpenSEA alliance to develop an open source 802.1x supplicant[25]. We will examine a number of these supplicants in more detail in the following paragraphs.

2.6.2 Available 802.1x supplicants

- Cisco Secure Services Client

The Cisco Secure Services Client (formerly Meetinghouse Data Communications Inc. AEGIS SecureConnect[26,27]) provides 802.1x supplicant authentication for access to both wired and wireless networks. It supports a wide range of operating systems, including: Windows, Linux, and MacOS. It also supports several EAP authentication methods including MD5, TLS, TTLS, and LEAP. In addition, as a part of the Cisco Unified Wireless Network, the Secure Services Client is claimed to provide the following benefits[28]:

(1) Simplify management of both wired and wireless networks
(2) Improve network security
(3) Reduce the total cost of network ownership

- Juniper Network Odyssey Access Client

Two of Funk Software's[29,30] well known products were Odyssey Access Client and Steel-Belted Radius. Odyssey Access Client is a 802.1x supplicant for wired and wireless networks. Steel-Belted Radius is an authentication server. Funk Software was acquired by Juniper Networks[31]. Juniper Network's Odyssey Access Client is an enterprise-class 802.1x supplicant software which supports WLAN security protocols very well. It is available for Windows 98/ME/2000/XP, and it supports MD5 and LEAP EAP authentication methods. Jupiter claims it has the following advantages[30]:

(1) It is a secure 802.1x supplicant for enterprises and government agencies.
(2) It makes use of the WPA2 protocol to protect credentials and network data on the wireless link.
(3) It reduces costs for enterprises.

- Microsoft 802.1x supplicant

Microsoft Windows 2000, Windows XP, and Windows Server 2003 families include built-in support for an IEEE 802.1x supplicant. This supplicant supports MD5 and TLS EAP authentication methods.

The most popular commercial 802.1x supplicants were introduced above. Next we will introduce some open source and free supplicants. There are three major open source supplicants: XSupplicant[32], wpa_supplicant[33], and Wire1x[34]. XSupplicant and wpa_supplicant are mainly used on Unix & UNIX like operating systems. While Wire1x is designed for various Windows platforms. We will describe each of these in further detail below.

- XSupplicant

XSupplicant is the outcome of the Open1X project[32]. The goal of the Open1X project was to develop an open source implementation of the IEEE 802.1x protocol.

This project focused on the development of a supplicant and authenticator. XSupplicant is an implementation of a 802.1x supplicant and it can be used in both wired and wireless LANs. XSupplicant makes use of a modular architecture so that new authentication methods can easily be added. Also, additional security components can be integrated into the system. The latest version of XSupplicant is 2.0.1 and it supports Linux/BSD and Apple Computer's Mac OS; and has a graphical user interface.

XSupplicant supports a variety of EAP authentication methods specifically:

- EAP-MD5
- LEAP
- EAP-MSCHAPv2
- EAP-AKA
- EAP-SIM
- EAP-TLS
- EAP-TTLS
- EAP-OTP
- EAP-PEAP (v0 and v1)

Based on the existing XSupplicant, an organization called the OpenSEA alliance was founded by six networking and security technology companies (Extreme Networks, Identity Engines, Infoblox, Symantec Corporation, TippingPoint, and Trapeze Networks) to pursue open source 802.1x supplicant development[25]. The name OpenSEA stands for Open Secure Edge Access. This alliance aims to develop an open source 802.1x supplicant which not only inherits the merits of XSupplicant, but also extends its functionality and supports additional platforms.

- Wpa_supplicant

Wpa_supplicant is a free IEEE 802.1x supplicant which supports a number of operating systems, including: Linux, BSD, Apple Computer's Mac OS X, and Microsoft's Windows operating system. It supports both WPA and WPA2 (IEEE 802.11i / RSN) [35]. Wpa_supplicant is designed to run in the background to control the wireless connection. Both a graphical user interface and a command line interface are available for monitoring the running supplicant. Via these user interfaces the user can see all the networks available via the computer's network interfaces. Wpa_supplicant supports following the WPA/IEEE 802.11i features[35]:

- WPA and full IEEE 802.11i/RSN/WPA2
- WPA with EAP (e.g., with a RADIUS authentication server)
- Key management for CCMP, TKIP, WEP
- WPA-PSK and WPA2-PSK (pre-shared key)
- RSN: PMKSA caching, pre-authentication

Wpa_supplicant also supports many EAP authentication methods, including[35]:

- EAP-AKA
- EAP-SIM
- EAP-PSK
- EAP-FAST
- EAP-PAX
- EAP-SAKE
- EAP-IKEv2
- EAP-TLS
- EAP-TTLS
- EAP-PEAP (both PEAPv0 and PEAPv1)
- LEAP (requires special support from the driver)

- WIRE1X

WIRE1X is another open source implementation of an IEEE 802.1x supplicant developed by the Wireless Internet Research & Engineering (WIRE) Laboratory in National Tsing Hua University in Taiwan[12]. Although the implementation of WIRE1X is based on Open1x, this supplicant was designed to run on various Microsoft Windows operating systems; so that users can gain access to the network more conveniently and securely than via Microsoft's own IEEE 802.1x supplicant. Currently, WIRE1X supports Windows Vista，Windows XP, Windows 2000, Windows ME, and Windows 98. The EAP authentication methods it supports include:

- AKA
- MD5
- SIM
- TLS
- TTLS
- PEAP
- MSCHAPv2

Compared with Microsoft's supplicant, WIRE1X supports more EAP authentication methods. In addition, the user interface of WIRE1X is very easy to use, thus a user can easily perform both installation and configuration. Therefore, WIRE1X is more practical and more widely used than Microsoft's supplicant.

Table 2-1 summarizes the differences between these supplicants. The development of a new supplicant in this thesis project builds upon this earlier work. Specifically it leverages the WIRE1x open source code.

Table 2-1. IEEE 802.1x Supplicants

Name	OS Supported	Supported EAP Methods	Source Availability
AEGIS SecureConnect	Windows/Linux/ MacOS	MD5, TLS, TTLS, LEAP	Non-open source
Odyssey Access Client	Windows 98/ME/ 2000/XP	MD5, LEAP	Non-open source
Microsoft Supplicant	Windows 2000/XP/ Server 2003	MD5, TLS	Non-open source
XSupplicant	Linux/BSD/MacOS	MD5,LEAP, MSCHAPV2, AKA, SIM, TLS, TTLS, OTP, PEAP	Open source
Wpa_supplicant	Linux/BSD/MacOS/ Windows	AKA, SIM, PSK, FAST,PAX, SAKE, IKEv2,TLS, TTLS, PEAP, LEAP	Open source
WIRE1X	Windows 98/ME/2000/XP	AKA, MD5, SIM, TLS, TTLS, PEAP, MSCHAPv2	Open Source

3. Analysis of non-binary authentication process

In the last chapter, we discussed the typical IEEE 802.1x authentication process which is a binary authentication approach. As shown in Figure 2.7, the supplicant can only send EAP type messages (such as EAP-Start and EAP-Response) before it receives an EAP-Success message. This means that before the supplicant is successfully authenticated, if it sends any other type of packets before it receiving an EAP-Success, then normally the authenticator will simply drop these other packets. However, a mobile user who moves into the coverage area of a new access point wishes to send and receive packets as soon as possible. Otherwise, their existing communication sessions (such as a VoIP session, file transfer, etc.) may be terminated due to the handover latency. In our project, a new non-binary authentication system tries to solve this problem. Potentially, the non-binary authenticator enables the supplicant to send a small number of packets and have them delivered to the correct destination - even before the supplicant has been authenticated. In this way, mobile users can continue to send VoIP packets **while** they are performing the authentication. An open question is when is the earliest that the non-binary access point can send an EAP-Success message. Another question is if the supplicant will listen to this EAP-Success message and continue to send other types of packets. We will address these questions and their answers in following sections. But first, it is necessary to introduce the concept of handover and the handover process in wireless networks.

3.1 Handover process in wireless networks

In wireless networks which use the architecture of the IEEE 802.11 standard, every supplicant is associated with an access point (AP) which provides access to the Internet. A supplicant may need to change the AP it is associated with when it moves from one area to another since the coverage area of each AP is limited. We make this change in APs via a handover process. As stated earlier, if a mobile user wants to maintain an ongoing conversation, then the duration for the handover process needs to be sufficiently short (as with a single IEEE 802.11 WLAN interface the interface can only be associated with a single AP at a time). We show the detailed handover process in wireless networks in Figure 3.1. (Note that in this figure we have assumed that the AP and the authenticator are co-located and label them both as “Authenticator” in the figure.)

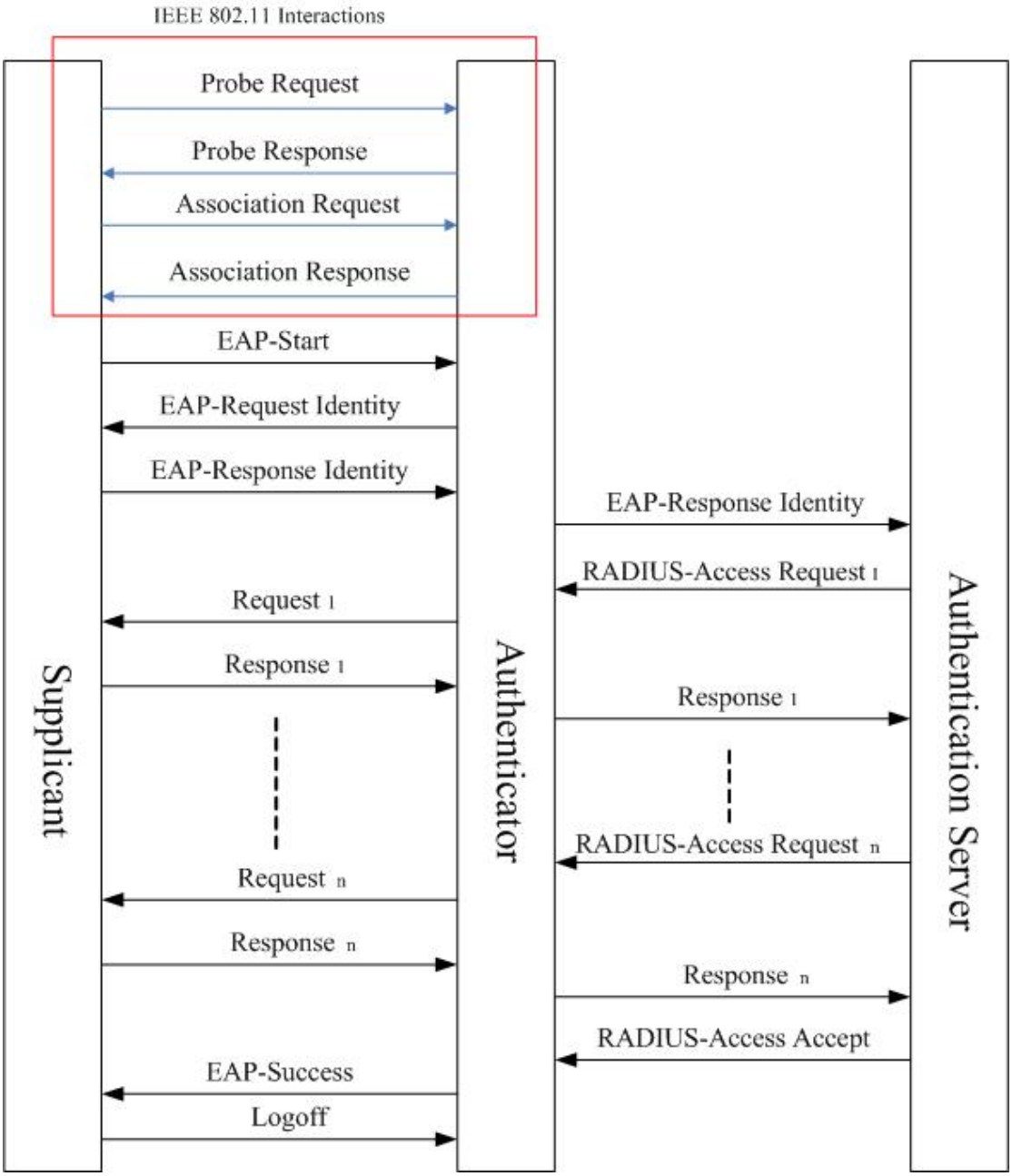


Figure 3.1 Handover process in wireless networks

During the first phase of the handover process the supplicant tries to probe for all potential new APs, in order to determine which AP it could potentially connect to if there is a need to change the AP. Based upon the Probe Response messages that the supplicant receives, it will decide which AP to associate with. The next phase is the association phase. The supplicant sends a 802.11 message to associate with the AP, and the AP will send an Association Response message in response. At this point a logical connection has been established between the supplicant and the AP. After the association phase, the supplicant sends an EAP-Start message to start the authentication phase. When the AP receives the identity information from the supplicant, it will forward this to the authentication server and the authentication

server will make a decision of whether the authenticator should be given permission to provide the supplicant with network access or not. Before the authentication server makes its final decision, there may be many messages exchange between the authentication server and the supplicant. Also, the time required to perform the authentication depends on the authentication method used. Note that the authenticator does not need to know which kind of authentication method is used - this is a matter between the supplicant and the authentication server. The authenticator's task is simply to forward Request messages from the authentication server to the supplicant and to forward Response messages from the supplicant to the authentication server. Therefore, the authenticator does not need to care about the contents of these messages, except for the RADIUS-Access Accept or RADIUS-Access Reject message from the authentication server - as the authenticator cares only about the decision (Success/Failure) and not the process or policy underlying this decision. After receiving the Success/Failure message, the authenticator forwards the EAP-Success or EAP-Failure message to the supplicant. It is important to note that the supplicant can be a good guy or a bad guy. According to the IEEE 802.1x model the supplicant is *assumed* to be a bad guy, thus this supplicant should *not* be permitted to send any packets other than EAP packets. If the supplicant is a good guy, then it will not send any other types of packets than EAP packets until it receives an EAP-Success message. But if it is a bad guy, it does not care about waiting for the EAP-Success message (as it will not receive such a message!) – However, it will try to sneak packets through the authenticator to the network. Unfortunately, for a good mobile user, his or her VoIP session may be terminated because of the relatively long authentication process. Therefore, a non-binary AP needs to send the EAP-Success message back as soon as possible so that the user's applications can continue to communicate with other hosts attached to the network. Otherwise the good guys are punished and the bad guys may or may not succeed in getting packets through to/from the network. How early can the supplicant receive an EAP-Success message? The following section will consider a number of possible scenarios.

3.2 When is the EAP-Success returned - several scenarios

The possible situations are shown in Figures 3.2 to 3.4. In Figure 3.2, after the association phase, the supplicant sets up a connection with the authenticator by sending an EAP-Start message. Once the authenticator is alerted by the supplicant, it will send an EAP-Request Identity message back to the supplicant. The supplicant sends an EAP-Response Identity message back to indicate its identity. During this process, the authentication server is not yet involved. At this point the authenticator can choose to send an EAP-Success message back to the supplicant, *assuming* that eventually this supplicant will be authenticated by the authentication sever - note that this is the opposite assumption made by an IEEE 802.1x authenticator. When the supplicant receives an EAP-Success message, it knows that it has the right to send all types of packets. However, as the authenticator does not yet know if the supplicant will be authenticated - if the supplicant attempts to send a lot of traffic, then the

non-binary AP can throw some of these packets away or delay them using a traffic shaper. Later, when the authenticator receives the actual EAP-Success message it can change the parameters of its traffic shaper/filter for this supplicant and send the supplicant another EAP-Success message. Note that for existing IEEE 802.1x supplicants the supplicant will simply ignore this message - as it has already received an EAP-Success message.

When the extended supplicant (as described in this thesis) receives the second EAP-Success message, it will know that it has been completely authenticated and can now send a lot of traffic. The advantage of this approach is that if most supplicants are going to be successfully authenticated, then there is no reason not to let them sent all types of packets. The disadvantages is that some supplicants will not be authenticated (i.e., they are bad guys), but they may be able to send some packets that would have been blocked using the IEEE 802.1x model. Thus the operator of the authenticator needs to decide if the risk of forwarding packets from a bad guy is worth the value of very low handoff delay for all of their actual subscribers.

In Figure 3.2, the non-binary Authenticator sends an EAP-Success message back to the supplicant immediately after it gets the supplicant's identity. Note that it could not have sent an EAP-Success earlier, as it needed the supplicant to identify itself. In the next scenario, the authenticator needs to trigger the supplicant to send the information which will be needed by the authentication server. In Figure 3.3, the non-binary authenticator in the AP sends an EAP-Success message back to the supplicant after it receives the EAP-Start message from the supplicant. Note that in both of these scenarios these messages are sent after the association phase. Since the supplicant has sent an EAP-Start message to the non-binary authenticator in the AP, the authenticator knows that the supplicant wants to trigger an authentication process. This is what a normal supplicant should do. However, for a sneaky supplicant (operated by a bad guy) once it finishes the association phase with the AP, it has a logical connection with the AP. Thus the supplicant can start to send other types of packets in the hope that they will be forwarded. Note that in the case of a good or bad guy, there is no *certainty* that packets are going to be forwarded until the supplicant receives an EAP-Success message. Thus even for the good guy it is necessary to send an EAP-Success message (as otherwise the supplicant might not try to send any other traffic). The advantage of this approach is that the supplicant will perform all the normal EAP interactions and if authenticated then the supplicant will experience only modest latency. The disadvantage is that a bad guy may be able to send some traffic before the authenticator learns from the authentication server that this supplicant should not be able to send any traffic.

In Figure 3.4, the non-binary authenticator in the AP sends an EAP-Success message back to the supplicant immediately after the association phase is over. This is the earliest time that the non-binary authenticator in the AP could send an EAP-Success message. Since the association phase is very short, the latency of the

handover can be reduced to a great extent. Although time is saved, the risk of sneaky supplicants is greatly increased. While this approach offers the lowest latency, there may be a problem as the supplicant might never send the authentication traffic - thus the authenticator will never be able to learn if this supplicant should get service or not.

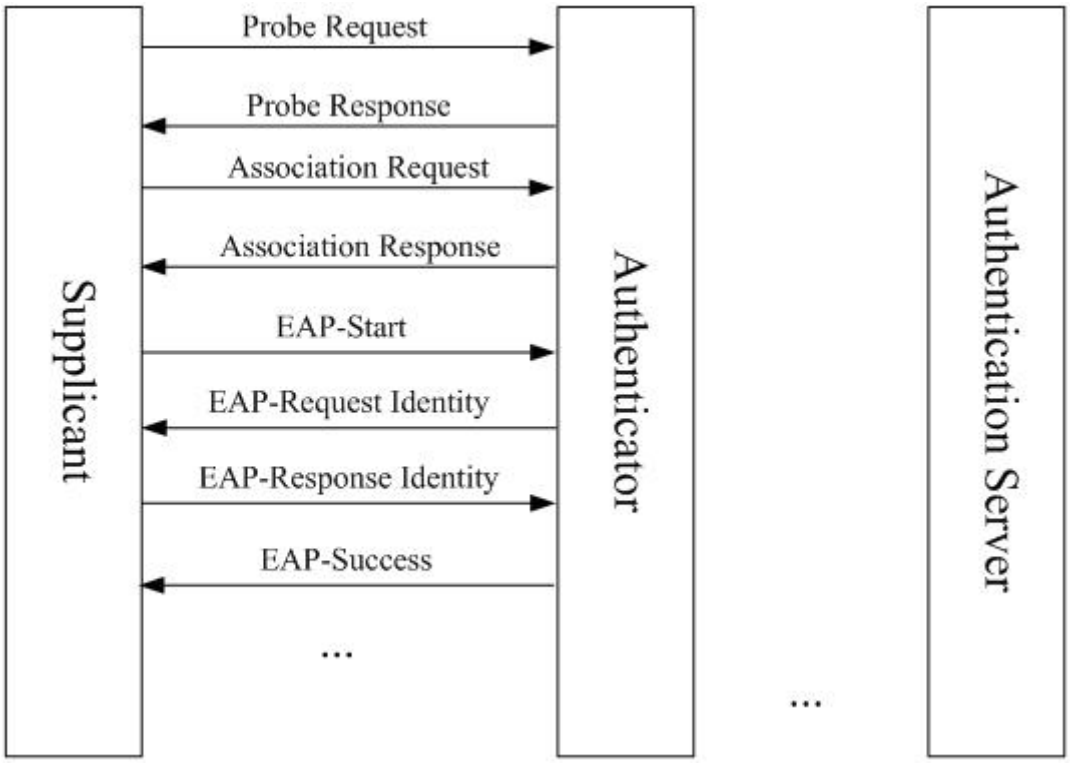


Figure 3.2 AP returns EAP-Success after getting a supplicant's identity

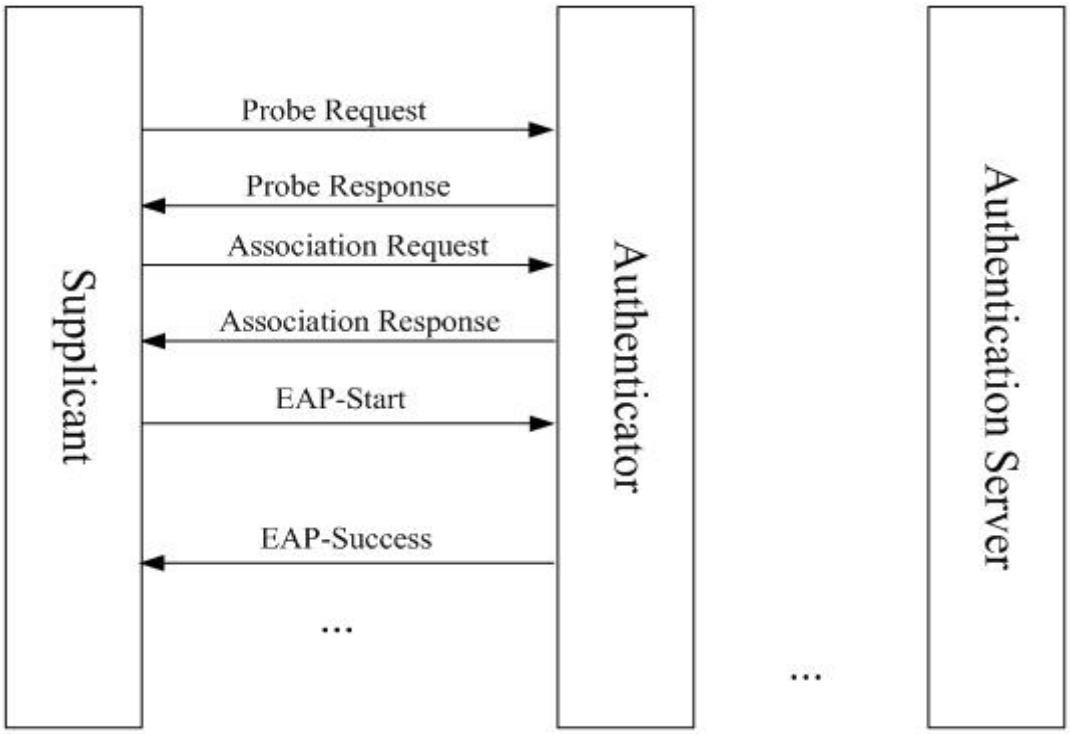


Figure 3.3 AP returns EAP-Success after getting EAP-Start message

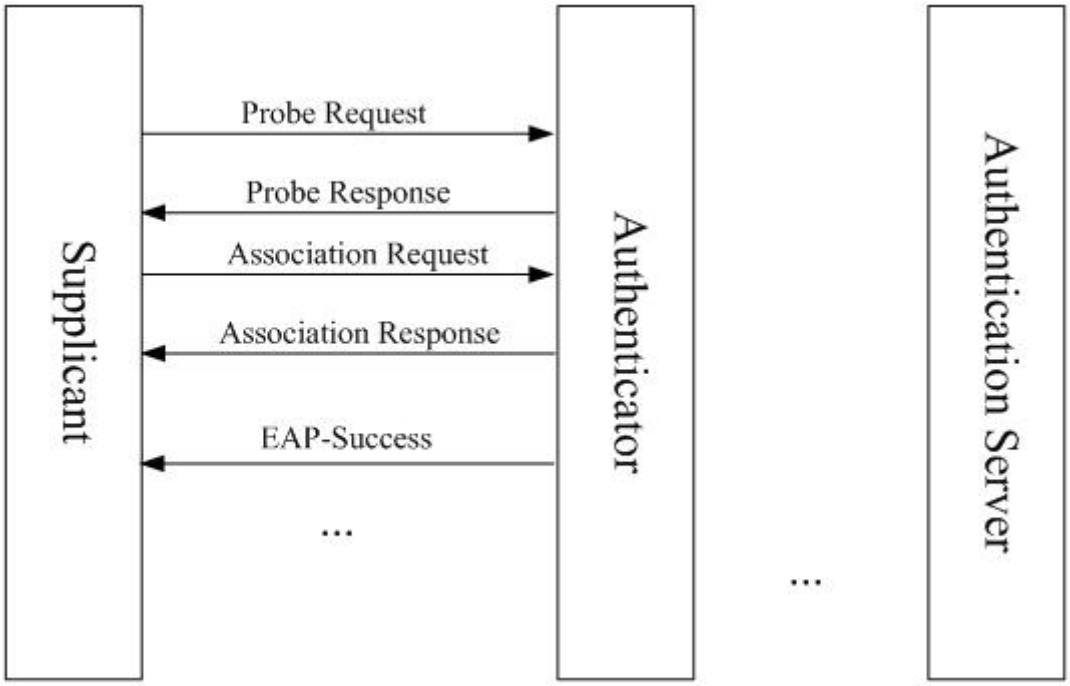


Figure 3.4 AP returns EAP-Success after association phase

Note that in these three scenarios the supplicant gets to send other types of traffic sooner than in the IEEE 802.1x case. Each of these scenarios represents different risk versus reward potentials. Also note that in the first case, if the authentication method used carries the necessary information in the first response from the supplicant, then the same supplicant can be used for both an IEEE 802.1x authenticator and a non-binary authenticator. Similarly in the second approach, a standard IEEE 802.1x supplicant can be used - but now with a full range of authentication methods. In the third approach a new supplicant is needed, as from the supplicant's point of view this authenticator does not exist and the access point simply looks like an open access point.

4. Experiment Analysis

In this chapter, we examine the capability of a supplicant to send and receive packets. The experiment was necessary in order to understand how existing supplicants would behave when sent the EAP-Success message at different times. The experiment consists of examining the response of a supplicant in two cases (i.e., with different authenticators).

4.1 Measurement tool

In our experiment, we used Wireshark[36] to capture packets and analyze them. Wireshark is a free network protocol analyzer and packet sniffer which is widely used for network analysis & troubleshooting by the telecommunication and software industry, and educational institutions. Before June 2006, this software was called Ethereal. Its name was subsequently changed to Wireshark because of trademark issues[37].

4.2 Experiments

4.2.1 After the supplicant is authenticated by KTHOPEN

In this section, we first examine the capability of a supplicant to send and receive packets after it is authenticated by the authenticator for KTHOPEN. On the KTH campus there is a wireless network in lecture rooms, labs, and public areas. This network is called KTHOPEN and there are many base stations covering most of the campus[38]. You can connect to the KTHOPEN network via a (built-in or plug-in) WiFi interface. To access the network via KTHOPEN, you need a KTH identity (provided either via a personal key or by entering your username and password via a web page - for the purpose of this thesis we will only discuss the later form of user authentication). In our experiment, we enabled a laptop's built-in wireless network interface, chose KTHOPEN as the wireless network to associate with, and clicked "OK" to complete the connection process. We started Internet Explorer as the web browser and entered the URL **www.kth.se**. Attempting to browse to any page is trapped and the browser is redirected to the log in page automatically.

After we entered a correct username and password for a KTH account, a "Login OK" page was shown. This indicates that the supplicant was authenticated by KTHOPEN. This process is shown in Figure 4.1 and Figure 4.2. The interaction process of the authentication is shown in Figure 4.3.

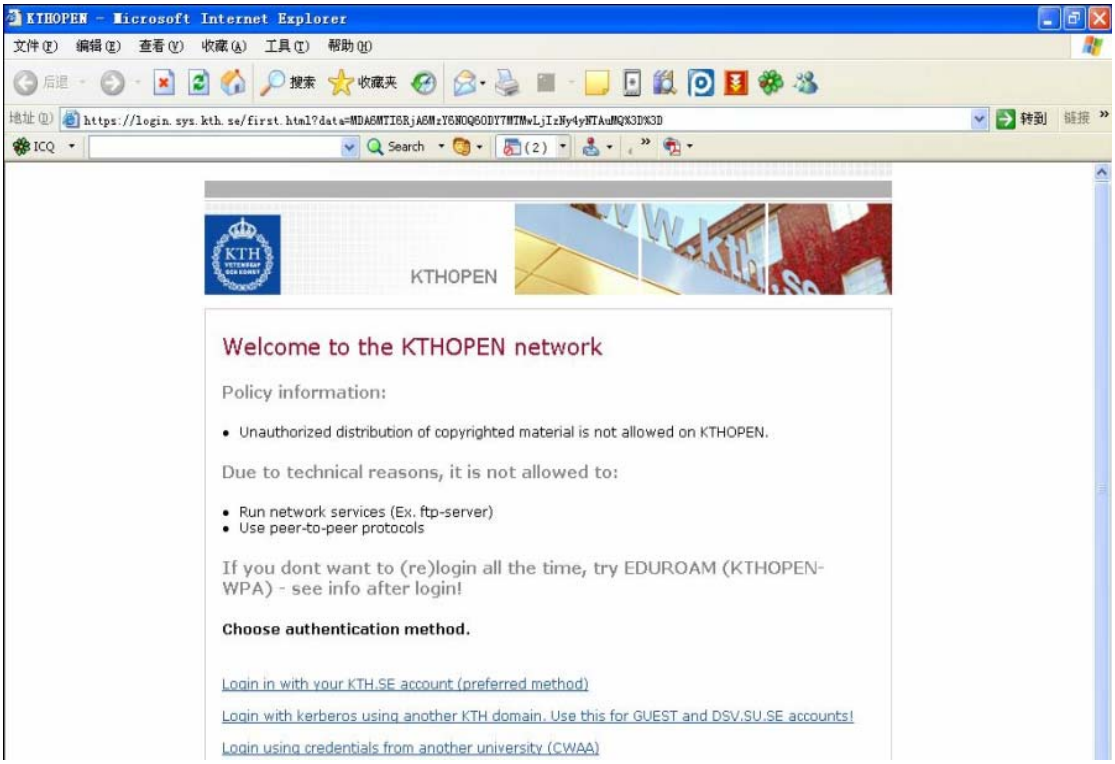


Figure 4.1 Log in page

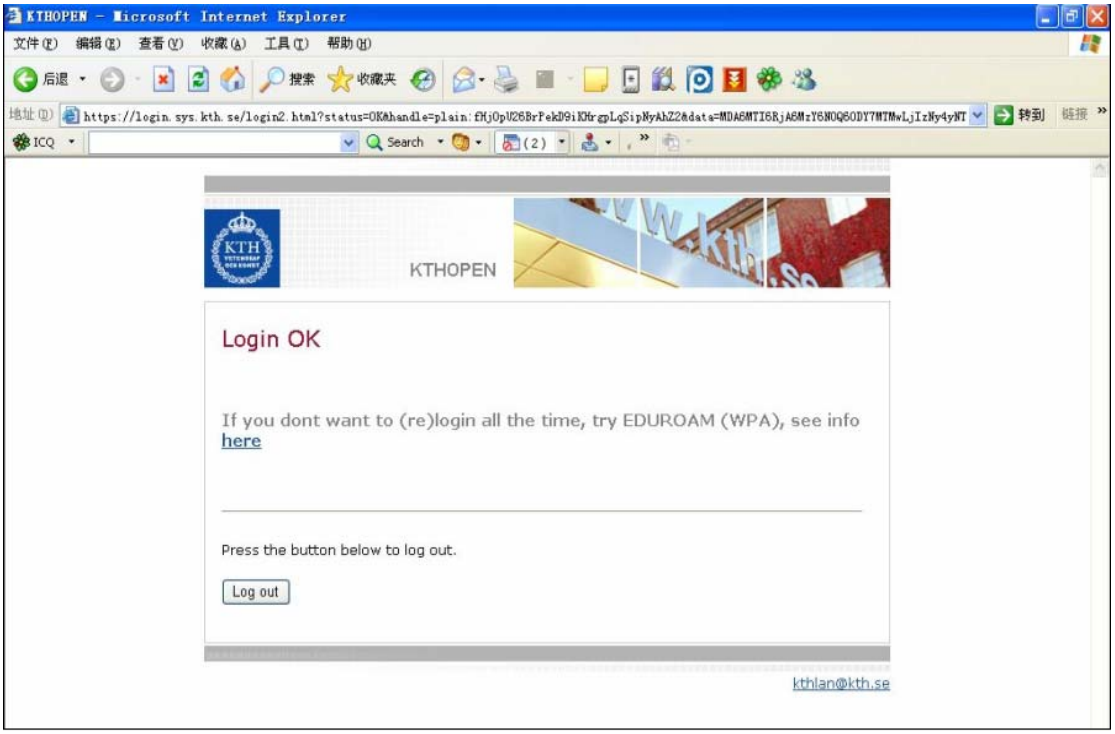
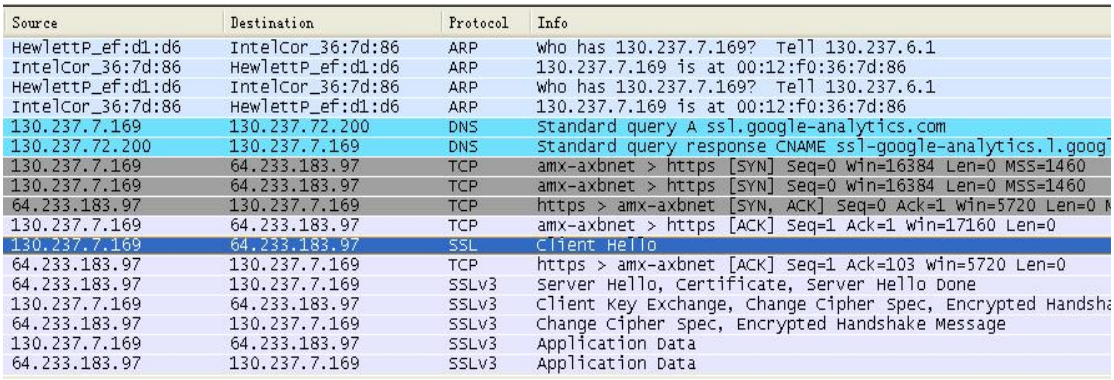


Figure 4.2 Log in OK page

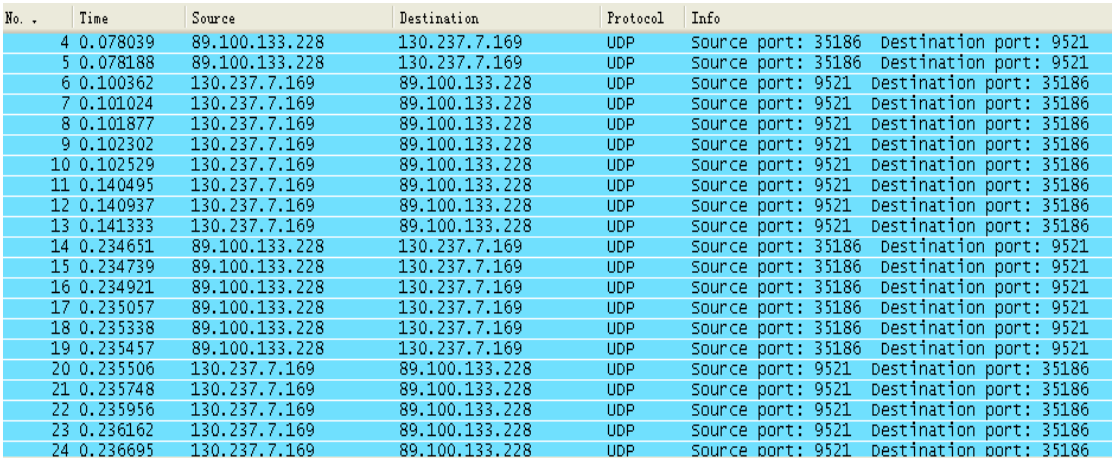


Source	Destination	Protocol	Info
HewlettP_ef:d1:d6	IntelCor_36:7d:86	ARP	Who has 130.237.7.169? Tell 130.237.6.1
IntelCor_36:7d:86	HewlettP_ef:d1:d6	ARP	130.237.7.169 is at 00:12:f0:36:7d:86
HewlettP_ef:d1:d6	IntelCor_36:7d:86	ARP	Who has 130.237.7.169? Tell 130.237.6.1
IntelCor_36:7d:86	HewlettP_ef:d1:d6	ARP	130.237.7.169 is at 00:12:f0:36:7d:86
130.237.7.169	130.237.72.200	DNS	Standard query A ssl.google-analytics.com
130.237.72.200	130.237.7.169	DNS	Standard query response CNAME ssl-google-analytics.l.goog
130.237.7.169	64.233.183.97	TCP	amx-axbnet > https [SYN] Seq=0 Win=16384 Len=0 MSS=1460
130.237.7.169	64.233.183.97	TCP	amx-axbnet > https [SYN] Seq=0 Win=16384 Len=0 MSS=1460
64.233.183.97	130.237.7.169	TCP	https > amx-axbnet [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 M
130.237.7.169	64.233.183.97	TCP	amx-axbnet > https [ACK] Seq=1 Ack=1 Win=17160 Len=0
130.237.7.169	64.233.183.97	SSL	Client Hello
64.233.183.97	130.237.7.169	TCP	https > amx-axbnet [ACK] Seq=1 Ack=103 Win=5720 Len=0
64.233.183.97	130.237.7.169	SSLv3	Server Hello, Certificate, Server Hello Done
130.237.7.169	64.233.183.97	SSLv3	Client Key Exchange, Change Cipher Spec, Encrypted Handsha
64.233.183.97	130.237.7.169	SSLv3	Change Cipher Spec, Encrypted Handshake Message
130.237.7.169	64.233.183.97	SSLv3	Application Data
64.233.183.97	130.237.7.169	SSLv3	Application Data

Figure 4.3 SSLv3 / TLS authentication

As shown in Figure 4.3, Secure Sockets Layer protocol version 3.0 / Transport Layer Security protocol (SSLv3 / TLS)[42] was used during the authentication process. The supplicant first sent a Client Hello message to the server. The server responded with a Server Hello message, as well as a Certificate message and a Sever Hello Done message. This indicated the handshake negotiation was finished. Then the supplicant responded with a Client Key Exchange message and a Change Cipher Spec record to tell the server that all the information will be encrypted. Finally, the supplicant sent an Encrypted Handshake message. The server sent a Change Cipher Spec and Encrypted Handshake message. The handshake was complete and the application protocol was enabled by then. Note that before the SSL/TLS handshake has even taken place, there was traffic going to IP address 64.233.183.97. It indicated that the network did not block all traffic other than the traffic required do authenticate and authorize the supplicant. Therefore the experiments with regard to KTHOPEN have nothing to do with IEEE 802.1x.

Wireshark also captured packets after the supplicant was authenticated by KTHOPEN (See Figure 4.4).



No.	Time	Source	Destination	Protocol	Info
4	0.078039	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
5	0.078188	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
6	0.100362	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
7	0.101024	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
8	0.101877	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
9	0.102302	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
10	0.102529	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
11	0.140495	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
12	0.140937	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
13	0.141333	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
14	0.234651	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
15	0.234739	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
16	0.234921	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
17	0.235057	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
18	0.235338	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
19	0.235457	89.100.133.228	130.237.7.169	UDP	Source port: 35186 Destination port: 9521
20	0.235506	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
21	0.235748	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
22	0.235956	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
23	0.236162	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186
24	0.236695	130.237.7.169	89.100.133.228	UDP	Source port: 9521 Destination port: 35186

Figure 4.4 Packets captured after the supplicant is authenticated by KTHOPEN

As shown in Figure 4.4, the supplicant's IP address was still 130.237.7.169. One of the destination addresses is 89.100.133.228. The supplicant could send UDP

packets to the destination and receive UDP packet from the destination as well. This proves that after the supplicant is authenticated by KTHOPEN, it can send and receive packets freely.

4.2.2 Before the supplicant is authenticated by a D-Link access point

A D-Link access point model DWL-G700 was used as both an access point and as an IEEE 802.1x authenticator. In this section and section 4.2.3, we will determine if the supplicant can send and receive packets before and after it is authenticated by the D-Link authenticator. For these experiment results, we setup two laptops and a D-Link access point. On one laptop, we installed and configured a WIRE1x supplicant running on Microsoft's Windows operating system. On the other laptop, we installed openSUSE running in a virtual machine. This operating system was used as the host operating system for our RADIUS authentication server. We also used this second laptop to configure and control the D-Link. The authenticator's IP address was 192.168.1.1. The authentication server's IP address was 192.168.1.5.

After installation of the WIRE1x supplicant program, when we first ran it, WIRE1x chose a wireless interface automatically and asked us if the selected interface was the one we want to use. If not, the user could choose the interface manually. The selected interface is stored for future use. See Figure 4.5 and Figure 4.6.

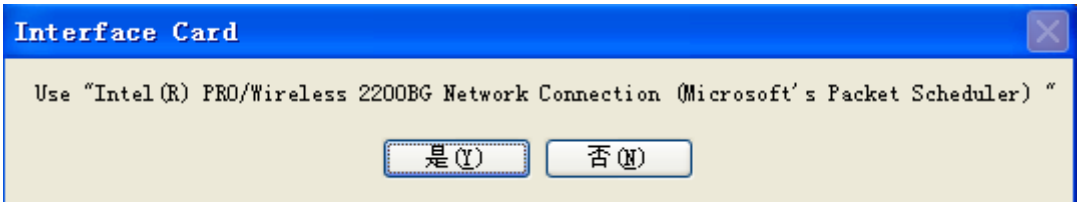


Figure 4.5 WIRE1x automatically chose a wireless interface

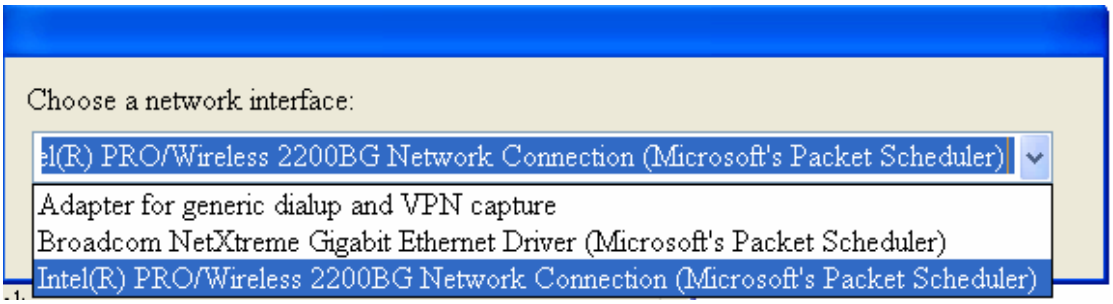


Figure 4.6 Choose the WiFi network interface

After choosing a wireless interface, we clicked "scan" to view all available wireless networks. All of the available wireless networks were shown in the list. The

user can also click the “refresh” button to cause the program to scan for available networks again. Then we chose our target AP (which was configured to have the SSID “ROOM 12”) and clicked “Associate” button (See Figure 4.7).

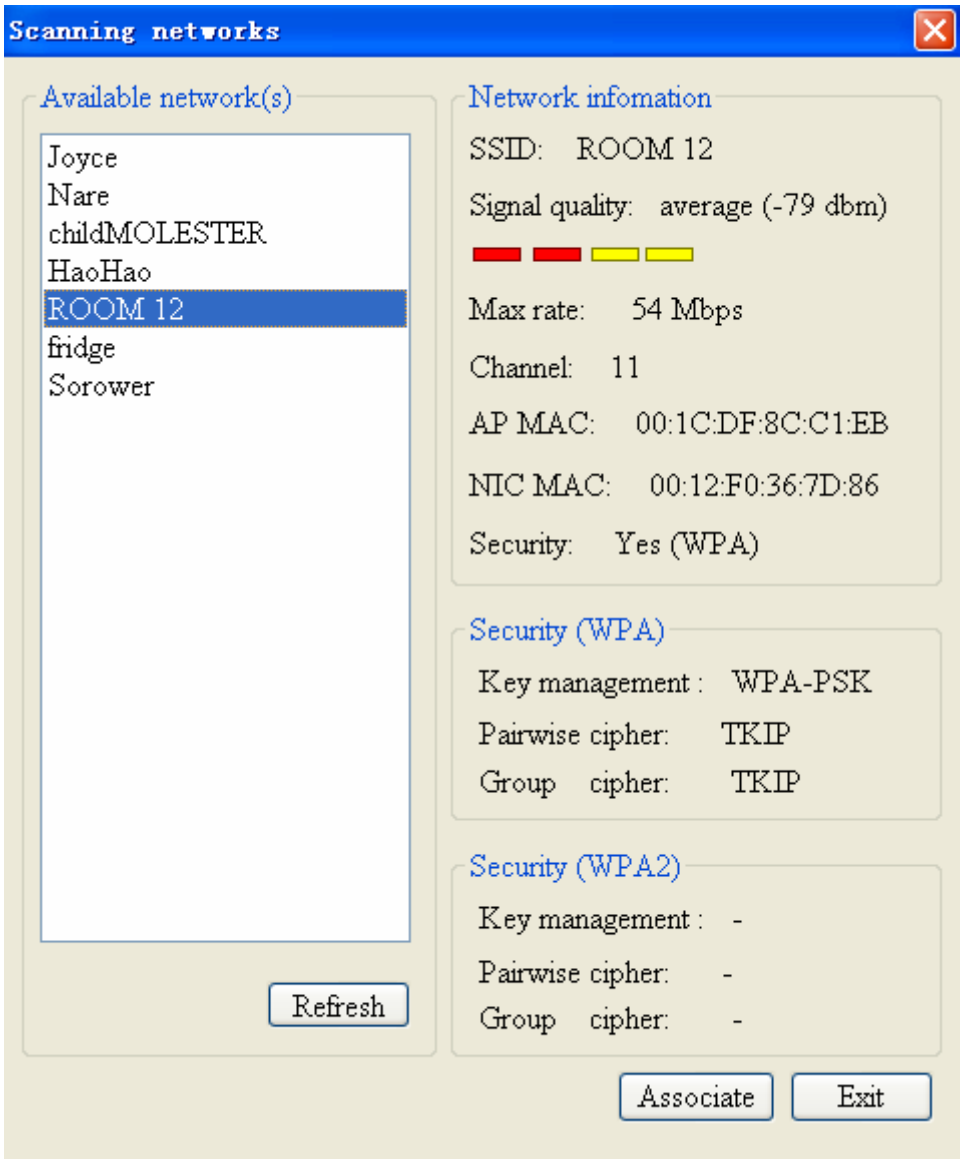


Figure 4.7 Scanning available wireless networks

Then we configured profile name and SSID and clicked “OK” button. After editing the profile, click “Save” button, see Figure 4.8 and Figure 4.9.

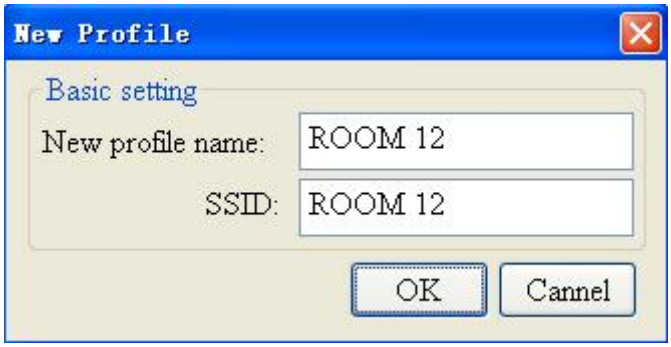


Figure 4.8 Configuration of profile name and SSID

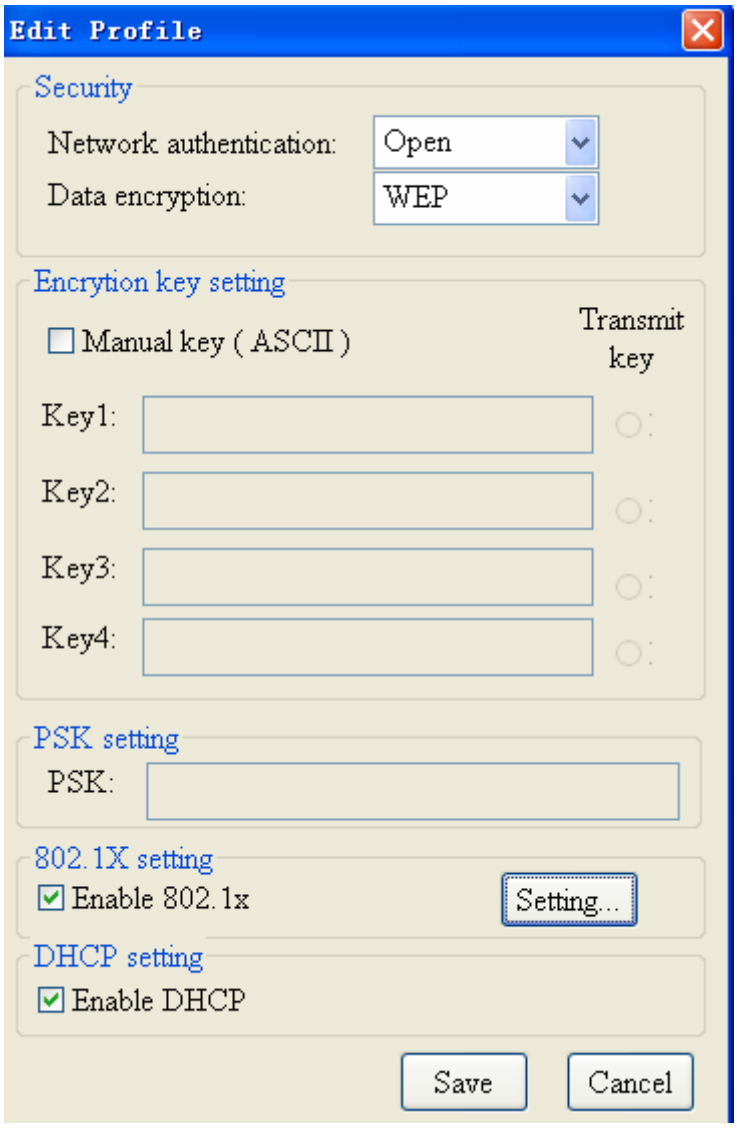


Figure 4.9 Editing profile

Finally, we started Wireshark, then clicked the “Associate” button and used Wireshark to monitor the interaction process between the supplicant and the authenticator. The result is shown in Table 4-1.

Table 4-1 Interaction between the supplicant and the authenticator

No.	Time	Source	Destination	Protocol	Info
1	0.000000	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
2	0.004502	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
3	2.712015	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xe843efc0
4	7.708180	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xe843efc0
5	16.707877	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xe843efc0
6	33.708156	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xe843efc0
7	54.138881	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
8	54.140452	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
9	57.710477	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb1834b94
10	62.708707	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb1834b94
11	71.708859	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb1834b94
12	88.709111	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb1834b94
..	... ...	... ...	... ...	...	... ...
..	... ...	... ...	... ...	...	... ...
..	... ...	... ...	... ...	...	... ...

As shown in Table 4-1, “IntelCor_36:7d:86” and “D-Link_f4:c5:e7” are the MAC address of the supplicant and the authenticator respectively. After the probe phase and association phase, the supplicant sent an EAP-Start message to the authenticator and the authenticator replied with an EAP-Request Identity message. Then the supplicant started to send DHCP requests to get an IP address. Note that, the supplicant repeated the DHCP request many times. But there was no reply from any DHCP server. This shows that if the authentication information is not configured correctly, then the supplicant will not be authenticated and that it could not successfully send any IP packets as it did not yet have a valid IP address. Additionally, the supplicant can not receive any packets since there is not yet a binding between the supplicant’s MAC address and a valid IP address.

To enable the supplicant to send UDP packets, we manually assigned an IP address (192.168.0.120) to the supplicant. See the resulting traffic in Table 4-2.

Table 4-2 Assign IP address to the supplicant manually

No.	Time	Source	Destination	Protocol	Info
1	0.000000	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
2	0.004502	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
3	3.656211	IntelCor_36:7d:86	Broadcast	ARP	Gratuitous ARP for 192.168.0.120 (Request)
4	4.463015	IntelCor_36:7d:86	Broadcast	ARP	Gratuitous ARP for 192.168.0.120 (Request)
5	5.463042	IntelCor_36:7d:86	Broadcast	ARP	Gratuitous ARP for 192.168.0.120 (Request)
6	6.496467	192.168.0.120	224.0.0.22	IGMP	V3 Membership Report / Join group 239.255.255.250 for any sources
7	6.504335	192.168.0.120	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
8	6.541383	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<00>
9	7.025580	192.168.0.120	224.0.0.22	IGMP	V3 Membership Report / Join group 239.255.255.250 for any sources
10	7.291241	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<00>
11	8.041256	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<00>
12	8.791268	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<00>
13	9.510224	192.168.0.120	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
14	9.573444	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<00>
15	10.322542	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<00>
16	11.072558	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<00>
17	11.822604	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<00>
18	12.525986	192.168.0.120	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
19	12.604322	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<20>
20	12.607041	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<1e>
21	13.353846	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<20>
22	13.353966	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<1e>
23	14.103861	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<20>
24	14.103979	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<1e>
25	14.853873	192.168.0.120	192.168.0.255	NBNS	Registration NB D8RRFP1X<20>
26	14.853997	192.168.0.120	192.168.0.255	NBNS	Registration NB MSHOME<1e>
27	15.636425	192.168.0.120	192.168.0.255	BROWSER	Request Announcement D8RRFP1X
28	15.639112	192.168.0.120	192.168.0.255	BROWSER	Host Announcement D8RRFP1X, Workstation, Server, NT Workstation, Potential Browser
29	16.141063	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
30	16.142796	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
31	17.135188	192.168.0.120	192.168.0.255	BROWSER	Request Announcement D8RRFP1X
..	... ...	... ...	... ...	...	... ...
..	... ...	... ...	... ...	...	... ...

As shown in Table 4-2, after we manually assigned an IP address to the supplicant, the supplicant attempted to send UDP packets despite *not being authenticated*. However, the authentication process did not continue after the EAPOL start and EAP Request (as the authentication server was not yet configured). The lack of responses to

these messages shows that before the authentication is successfully finished, the UDP packets could not be successfully delivered to their destinations. To examine the supplicant's capability to receive traffic prior to the authentication, we used another host and made it to send traffic to the supplicant. The result is shown in Table 4-3.

Table 4-3 Send traffic to the supplicant from another host

Source	Destination	Protocol	Info
89.180.161.143	130.237.239.78	UDP	Source port: 12398 Destination port: 9521
130.237.239.78	89.180.161.143	UDP	Source port: 9521 Destination port: 12398
89.180.161.143	130.237.239.78	UDP	Source port: 12398 Destination port: 9521
130.237.239.78	89.180.161.143	UDP	Source port: 9521 Destination port: 12398
130.237.239.78	192.168.0.120	ICMP	Echo (ping) request
89.180.161.143	130.237.239.78	UDP	Source port: 12398 Destination port: 9521
130.237.239.78	89.180.161.143	UDP	Source port: 9521 Destination port: 12398
89.180.161.143	130.237.239.78	UDP	Source port: 12398 Destination port: 9521
130.237.239.78	89.180.161.143	UDP	Source port: 9521 Destination port: 12398
... ...	... ...	...	... ...
... ...	... ...	...	... ...
130.237.239.78	192.168.0.120	ICMP	Echo (ping) request
89.180.161.143	130.237.239.78	UDP	Source port: 12398 Destination port: 9521
130.237.239.78	89.180.161.143	UDP	Source port: 9521 Destination port: 12398
89.180.161.143	130.237.239.78	UDP	Source port: 12398 Destination port: 9521
130.237.239.78	89.180.161.143	UDP	Source port: 9521 Destination port: 12398
... ...	... ...	...	... ...
... ...	... ...	...	... ...

As shown in Table 4-3, another host with IP address 130.237.239.78 sent Echo request to the supplicant from time to time. However, there was no response from the supplicant. Therefore before the authentication is successfully finished, the supplicant can not receive any traffic other than those which is used for authentication and authorization.

4.2.3 After the supplicant is authenticated by D-Link

In this section, the experimental environment is similar to that described in section 4.2.2. The difference is that we configured a specific type of authentication method and did some additional configuration of the supplicant. WIRE1x supports a number of authentication methods. We chose EAP-MD5 as the authentication method to be used. As shown in Figure 4.9, in section 4.2.2 we configured the WIRE1x supplicant, we first selected "Enable 802.1x", then clicked "Setting" button to enter the details of our configuration. The configuration processes are shown in Figure 4.10 and Figure 4.11.

Figure 4.10 Select authentication method

Figure 4.11 Set username and password

After the configurations were finished and saved, we clicked the “Associate” button to start the authentication process. The capture of the traffic is shown in Table 4-4.

Table 4-4 After the supplicant was authenticated

No.	Time	Source	Destination	Protocol	Info
1	0.000000	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
2	0.058493	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
3	0.077240	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAP	Response, Identity [RFC3748]
4	1.015882	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Auth [RFC3748]
5	1.052638	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAP	Response, Auth [RFC3748]
6	1.095311	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Success [RFC3748]
7	1.144147	0.0.0.0	255.255.255.255	DHCP	DHCP Discover – Transaction ID 0xb7a962df
8	1.148925	192.168.1.1	192.168.1.2	ICMP	Echo (ping) request
9	1.750938	192.168.1.1	192.168.1.2	DHCP	DHCP Offer – Transaction ID 0xb7a962df
10	1.775714	0.0.0.0	255.255.255.255	DHCP	DHCP Request – Transaction ID 0xb7a962df
11	1.811022	192.168.1.1	192.168.1.2	DHCP	DHCP ACK – Transaction ID 0xb7a962df
12	3.204338	D-Link_f4:c5:e7	IntelCor_36:7d:86	ARP	Who has 192.168.1.2? Tell 192.168.1.1
13	3.204869	IntelCor_36:7d:86	D-Link_f4:c5:e7	ARP	192.168.1.2 is at 00:12:f0:36:7d:86
14	6.043206	192.168.1.2	60.28.213.176	UDP	Source port: 9521 Destination port: 17788
15	6.043451	192.168.1.2	61.172.196.109	UDP	Source port: 9521 Destination port: 17788
16	6.043536	192.168.1.2	121.9.201.99	UDP	Source port: 9521 Destination port: 17788
17	6.043660	192.168.1.2	125.46.57.14	UDP	Source port: 9521 Destination port: 17788
18	7.017898	60.28.213.176	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
19	7.018366	61.172.196.109	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
20	7.018519	121.9.201.99	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
21	7.019285	125.46.57.14	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
..	... ...	... ...	... ...	...	... ...
..	... ...	... ...	... ...	...	... ...

As shown in Table 4-4, after the authenticator received the supplicant’s identity and authentication information, it returned an EAP-Success message to the supplicant. Then the supplicant sent a DHCP request and received response with an assigned IP address. After that the supplicant could send UDP packets to their destinations and receive packets from destinations as well. This experiment showed that after the supplicant was correctly configured and authenticated by the D-Link authenticator, that the supplicant could send and receive UDP packets freely.

5. Implementation, testing, and analysis of a new supplicant

5.1 Implementation of a new supplicant

As we mentioned in the goals of this thesis, this thesis project mainly focuses on the supplicant and tries to test the authenticator by using a new supplicant. The original supplicant program of WIRE1x only sends EAP authentication packets. Based on the WIRE1x code, a new supplicant was developed that can send UDP packets in addition to the EAP-Start packets. These additional UDP packets are used to determine if and when non EAPOL packets can be sent by the supplicant. In order to do this, the “eapol.cpp” code was modified in order to construct and send additional UDP packets. To implement this, a new function “txSendUdp” was added, and this function was embedded in the “txStart” function. The “txStart” function was defined to send EAP-Start packets. After EAP-Start packets were sent, the function “txSendUdp” was invoked to send UDP packets. The main portions of the new added code are listed here.

```
/*********************************************************************************************
*    Name: Make udp Package
*    Created: Hengchong
*********************************************************************************************/
int MakeUdp(u_char * udpPackage)
{
     //Construct space to store packets
     u_char *udppackage;
     u_char *src_addr;
     u_char domain[50];//domain name
     DLC_Header *dlcheader,*fakedlc;
     IpHeader *ipheader,*fakeip;//IP header
     UDP_Header *fakeudp;//UDP header
     DNS_HEADER *fakedns;//DNS header
     PSD_HEADER *psdheader;
//   USHORT ip_len;//IP header length
     char srcip[4];//Source IP address
     char destip[4];//Destination IP address
     char srcmac[6];//Source MAC address
//   char destmac[6];// Destination MAC address
//   char domain[50];
     //Fake IP address infomation
     memset(srcip,0,4);
     memset(destip,0,4);
```

```
//Construct fake respondent packets
fakedns=(DNS_HEADER *)malloc(12+strlen((char *)domain)+21);//Assign memory for DNS packets
    memset(fakedns,0,12+strlen((char *)domain)+21);
    fakedns->id=2222;
    fakedns->flags=htons(0x8180);
    fakedns->questions=htons(1);
    fakedns->answers=htons(1);
    fakedns->author=0;
    fakedns->addition=0;
    memcpy((char *)fakedns+12,domain,strlen((char *)domain)+1);
    *((BYTE *)fakedns+12+strlen((char *)domain)+1)=0x00;//Type query
    *((BYTE *)fakedns+12+strlen((char *)domain)+2)=0x01;
    *((BYTE *)fakedns+12+strlen((char *)domain)+3)=0x00;
    *((BYTE *)fakedns+12+strlen((char *)domain)+4)=0x01;
    *((BYTE *)fakedns+12+strlen((char *)domain)+5)=0xC0;
    *((BYTE *)fakedns+12+strlen((char *)domain)+6)=0x0C;
    *((BYTE *)fakedns+12+strlen((char *)domain)+7)=0x00;
    *((BYTE *)fakedns+12+strlen((char *)domain)+8)=0x01;
    *((BYTE *)fakedns+12+strlen((char *)domain)+9)=0x00;
    *((BYTE *)fakedns+12+strlen((char *)domain)+10)=0x01;
    *((BYTE *)fakedns+12+strlen((char *)domain)+11)=0x00;//TTL
    *((BYTE *)fakedns+12+strlen((char *)domain)+12)=0x00;
    *((BYTE *)fakedns+12+strlen((char *)domain)+13)=0x00;
    *((BYTE *)fakedns+12+strlen((char *)domain)+14)=0x80;
    *((BYTE *)fakedns+12+strlen((char *)domain)+15)=0x00;//IP address length
    *((BYTE *)fakedns+12+strlen((char *)domain)+16)=0x04;
    //Fill fake IP address
    *((u_char *)fakedns+12+strlen((char *)domain)+17)=192;
    *((u_char *)fakedns+12+strlen((char *)domain)+18)=168;
    *((u_char *)fakedns+12+strlen((char *)domain)+19)=1;
    *((u_char *)fakedns+12+strlen((char *)domain)+20)=2;
    //Fill udp packets
    fakeudp=(UDP_Header *)malloc(8+12+strlen((char *)domain)+21);
    memset((char *)fakeudp,0,8+12+strlen((char *)domain)+21);
    fakeudp->DstPort=7000;
    fakeudp->SrcPort=6000;
    fakeudp->Chksum=0;//Check sum
    fakeudp->Len=htons(8+12+strlen((char *)domain)+21);//UDP packets size
    memcpy((char *)fakeudp+8,(char *)fakedns,12+strlen((char *)domain)+21);
    psdheader=(PSD_HEADER *)malloc(12+8+12+strlen((char *)domain)+21);
    psdheader->mbz=0;
    psdheader->ptcl=17;
    psdheader->udpl=htons(8+12+strlen((char *)domain)+21);
    memcpy((char *)&(psdheader->saddr[0]),destip,4);
```

```
        memcpy((char *)&(psdheader->daddr[0]),srcip,4);
        memcpy((char *)psdheader+12,(char *)fakeudp,8+12+strlen((char *)domain)+21);
        fakeudp->Chksum=checksum((USHORT *)psdheader,12+8+12+strlen((char *)domain)+21);
        //Fill IP packets
        fakeip=(IpHeader *)malloc(20+8+12+strlen((char *)domain)+21);
        memset((char *)fakeip,0,20+8+12+strlen((char *)domain)+21);
        fakeip->h_len=0x45;
        fakeip->checksum=0;
        fakeip->tos=ipheader->tos;
        fakeip->total_len=htons(20+8+12+strlen((char *)domain)+21);
        fakeip->ident=ipheader->ident;//Identity
        fakeip->frag_and_flags=ipheader->frag_and_flags;
        fakeip->proto=ipheader->proto;
        fakeip->ttl=128;
        memcpy((char *)fakeip+12,(char *)destip,4);//Source IP address
        memcpy((char *)fakeip+16,(char *)srcip,4);//Destination IP address
        memcpy((char *)fakeip+20,(char *)fakeudp,8+12+strlen((char *)domain)+21);
        //Calculate Checksum
        fakeip->checksum=checksum((USHORT *)fakeip,20);
        //Fill MAC frame
        fakedlc=(DLC_Header *)malloc(14+20+8+12+strlen((char *)domain)+21);
        memset(fakedlc,0,14+20+8+12+strlen((char *)domain)+21);
       src_addr = get_src_mac();
        memcpy((char *)&(fakedlc->DesMAC[0]),eapol_dst,6);//Fill destination MAC
        memcpy((char *)&(fakedlc->SrcMAC[0]),src_addr,6);//Fill source MAC
        free(src_addr);
        src_addr = NULL;
        fakedlc->Ethertype=dlcheader->Ethertype;
        memcpy((char *)fakedlc+14,(char *)fakeip,20+8+12+strlen((char *)domain)+21);
        udpPackage=(u_char *)fakedlc;
        return (14+20+8+12+strlen((char *)domain)+21);
}
/*********************************************************************************************
*    Name: SendUdpPackage
*    Description:
*    Created:Hengchong
*********************************************************************************************/
void txSendUdp()
{
    printf("send txStart\n");
    u_char *temp=NULL;
    int len = MakeUdp(temp);
    if (send_frame(temp, len) == NULL)
        printf("send udp to link.\n");
```

```
    free(temp);
    temp = NULL;
}
```

The complete source code is included as Appendix A. This new supplicant can send both EAP-Start messages and UDP packets. It is useful to either a good guy or a bad guy to find out that if the authenticator is an open access point or not. The former can continue his/her VoIP session or other activities in parallel with the authentication and authorization process. The later may try to sneak more packets and occupy bandwidth resources as much as possible.

5.2 Imitation of a non-binary AP

The new supplicant program was intended to test and stress the authenticator which was being implemented by Guo Jia. However, he has not yet finished this implementation. Therefore, a non-binary AP and supplicant needed to be emulated in order to complete the test and analysis for this thesis project. The general process to do this was: Start two processes on one laptop. One process is used for the supplicant and the other one is used for the authenticator. Assign different IP addresses for the supplicant and the authenticator, so that they can communicate with each other. In the case of an actual interaction between a supplicant and an IEEE 802.11 access point and IEEE 802.1x authenticator, after the probe and association phase, the supplicant would send an EAP-Start message to the authenticator to initiate the authentication process. Then, the authenticator sends an EAP-Request / Identity message back to the supplicant. Using the non-binary authenticator model, the authenticator also sends a bandwidth specification with the value M1 to the supplicant. As long as the bandwidth that the supplicant asks for is no greater than M1, the supplicant can access the network temporarily if its usage remains within this limited bandwidth even without being authenticated. The authentication process continues between the supplicant and the authenticator. If the supplicant is subsequently successfully authenticated, then the authenticator will send an EAP-Success message back, along with a new bandwidth value M2. The new bandwidth value M2 indicates the bandwidth allowed for the supplicant after it is authenticated. If the authentication is not successful, then the authenticator will send an EAP-Failure message back and close the limited bandwidth connection. The entire process is shown in Figure 5.1.

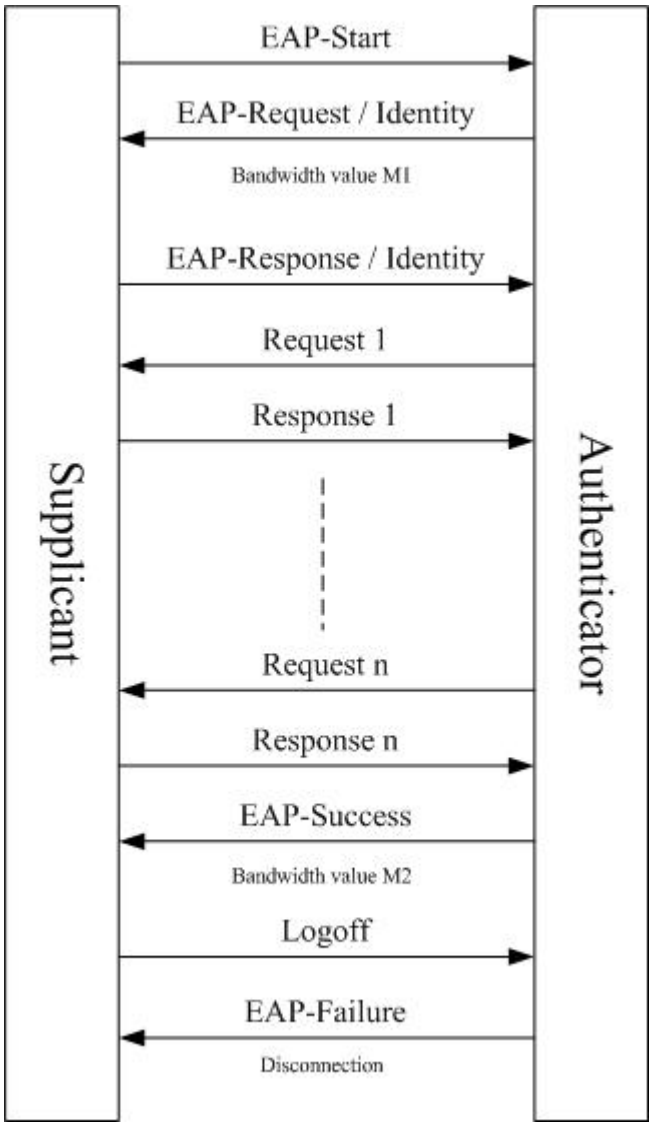


Figure 5.1 Non-binary authentication process

5.3 Test and Analysis

In this section, we test the new supplicant first. The experiment environment is similar to that described in section 4.2.3. In the case that before the new supplicant was authenticated by the D-link authenticator, we used Wireshark to monitor the interaction process between the new supplicant and the authenticator. The result is shown in Table 5-1.

Table 5-1 Interaction process between the supplicant and the authenticator
(before the supplicant was authenticated)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
2	0.001534	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
3	1.562611	IntelCor_36:7d:86	Broadcast	ARP	Gratuitous ARP for 192.168.1.2 (Request)
4	2.510364	IntelCor_36:7d:86	Broadcast	ARP	Gratuitous ARP for 192.168.1.2 (Request)
5	3.340358	IntelCor_36:7d:86	Broadcast	ARP	Gratuitous ARP for 192.168.1.2 (Request)
6	4.764649	192.168.1.2	224.0.0.22	IGMP	V3 Membership Report / Join Group 239.255.255.250 for any sources
7	4.808615	192.168.1.2	192.168.1.1	UDP	Source port: 6000 Destination port: 7000
8	6.392886	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
9	6.396192	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10	6.400635	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
11	6.403577	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
12	6.407512	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
13	20.862331	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
14	20.864508	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
15	20.981658	192.168.1.2	192.168.1.1	UDP	Source port: 6000 Destination port: 7000
16	22.862856	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
17	22.866522	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
..	... ...	... ...	... ...	...	... ...
..	... ...	... ...	... ...	...	... ...

As shown in Table 5-1, “IntelCor_36:7d:86” and “D-Link_f4:c5:e7” are the MAC address of the supplicant and the authenticator respectively. After the supplicant was assigned the IP address 192.168.1.2, it attempted to send an UDP packet despite *not being authenticated*. However, the authentication process did not continue after the EAPOL-Start and EAP-Request / Identity due to the unfinished configuration of the authentication server. The lack of response to this domain name query message shows that before the authentication is successfully finished, the UDP packet could not be successfully delivered to its destination.

In the case that after the new supplicant was authenticated by the D-link authenticator, we chose EAP-MD5 as the authentication method to be used. The result of the interaction process is shown in Table 5-2.

Table 5-2 Interaction process between the supplicant and the authenticator
(after the supplicant was authenticated)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAPOL	Start
2	0.001677	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Identity [RFC3748]
3	0.002052	192.168.1.2	192.168.1.1	UDP	Source port: 6000 Destination port: 7000
4	0.003521	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAP	Response, Identity [RFC3748]
5	0.004898	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Request, Auth [RFC3748]
6	0.005921	IntelCor_36:7d:86	D-Link_f4:c5:e7	EAP	Response, Auth [RFC3748]
7	0.007528	D-Link_f4:c5:e7	IntelCor_36:7d:86	EAP	Success [RFC3748]
8	0.009153	192.168.1.2	58.211.82.166	UDP	Source port: 9521 Destination port: 17788
9	0.009331	192.168.1.2	121.12.255.104	UDP	Source port: 9521 Destination port: 17788
10	0.009522	192.168.1.2	125.46.57.6	UDP	Source port: 9521 Destination port: 17788
11	0.009716	192.168.1.2	58.211.82.167	UDP	Source port: 9521 Destination port: 17788
12	0.009985	192.168.1.2	121.12.255.107	UDP	Source port: 9521 Destination port: 17788
13	0.010162	192.168.1.2	125.46.57.7	UDP	Source port: 9521 Destination port: 17788
14	1.002632	58.211.82.166	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
15	1.002788	121.12.255.104	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
16	1.002823	125.46.57.6	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
17	1.002977	58.211.82.167	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
18	1.003056	121.12.255.107	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
19	1.003127	125.46.57.7	192.168.1.2	UDP	Source port: 17788 Destination port: 9521
..	... ...	... ...	... ...	...	... ...
..	... ...	... ...	... ...	...	... ...

As shown in Table 5-2, before the supplicant was successfully authenticated, the supplicant sent an UDP packet to the destination but it did not get a response from the destination. After the authenticator received the supplicant's identity and authentication information, it returned an EAP-Success message to the supplicant. Then the supplicant could send UDP packets to their destinations and receive packets from destinations as well. This experiment showed that after the supplicant was correctly configured and authenticated by the D-Link authenticator, that the supplicant could send UDP packets and that these packets could be delivered to the destination.

In the text below we describe the testing of the emulation program. The emulation program integrated the supplicant and the authenticator together. The source code for this program is included in Appendix B. The initial interface state is shown in Figure 5.2.

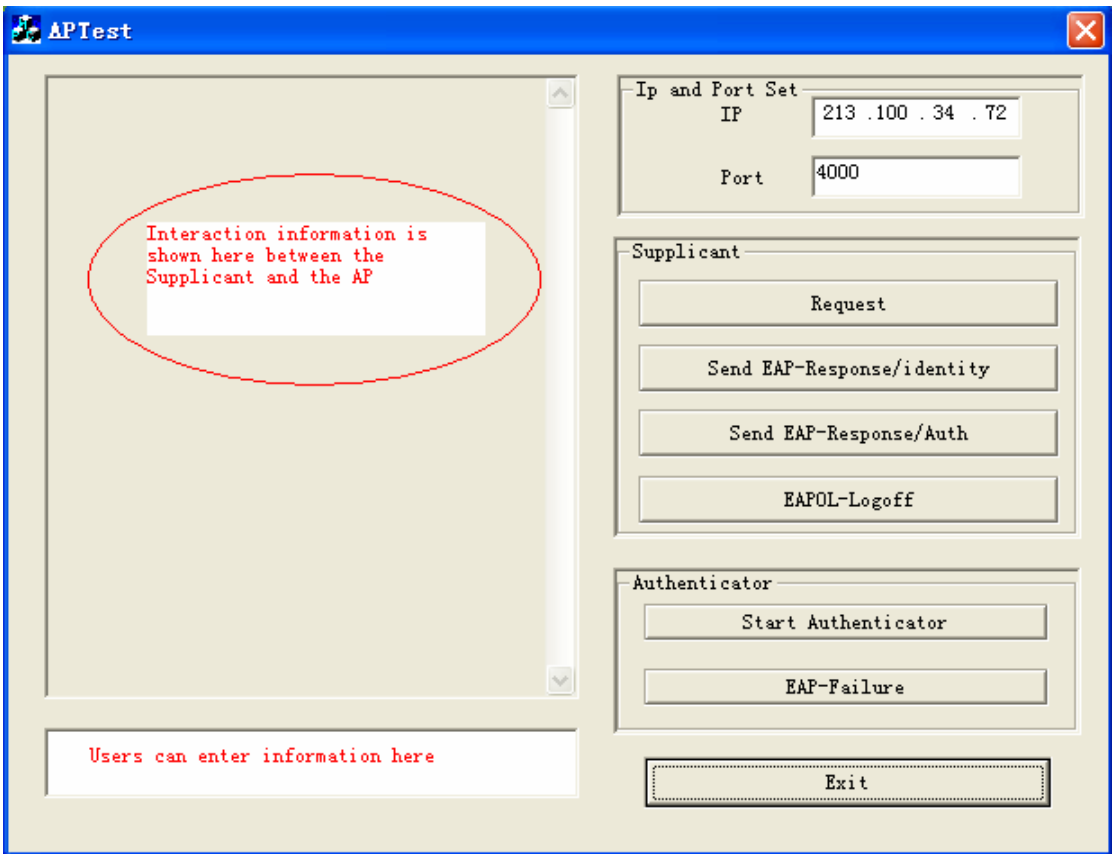


Figure 5.2 Initial interface of the emulation program

As shown in Figure 5.2, on the right there are four buttons which belong to the supplicant and two buttons which belong to the authenticator. This enables the user to manually invoke these separate operations. Additionally, the user can manually set the IP addresses of the supplicant and the authenticator. On the left, in the large field on the top, the interaction between the supplicant and the authenticator will be shown. When necessary users can enter information in the bottom left input box. For example, the user can manually enter the maximum bandwidth that the supplicant is assigned.

This program can be run on two laptops or as two programs on one laptop. In our test, we ran two such programs on one laptop. One program acts as the supplicant and the other one as the authenticator. We set the IP address 213.100.34.72 for the supplicant and 213.100.34.75 for the authenticator. In the authenticator, we clicked “Start Authenticator” button, then the result “Server has been set ok” was shown in the interaction information area. This test program allows us to emulate the non-binary authentication process (see Figure 5.3).

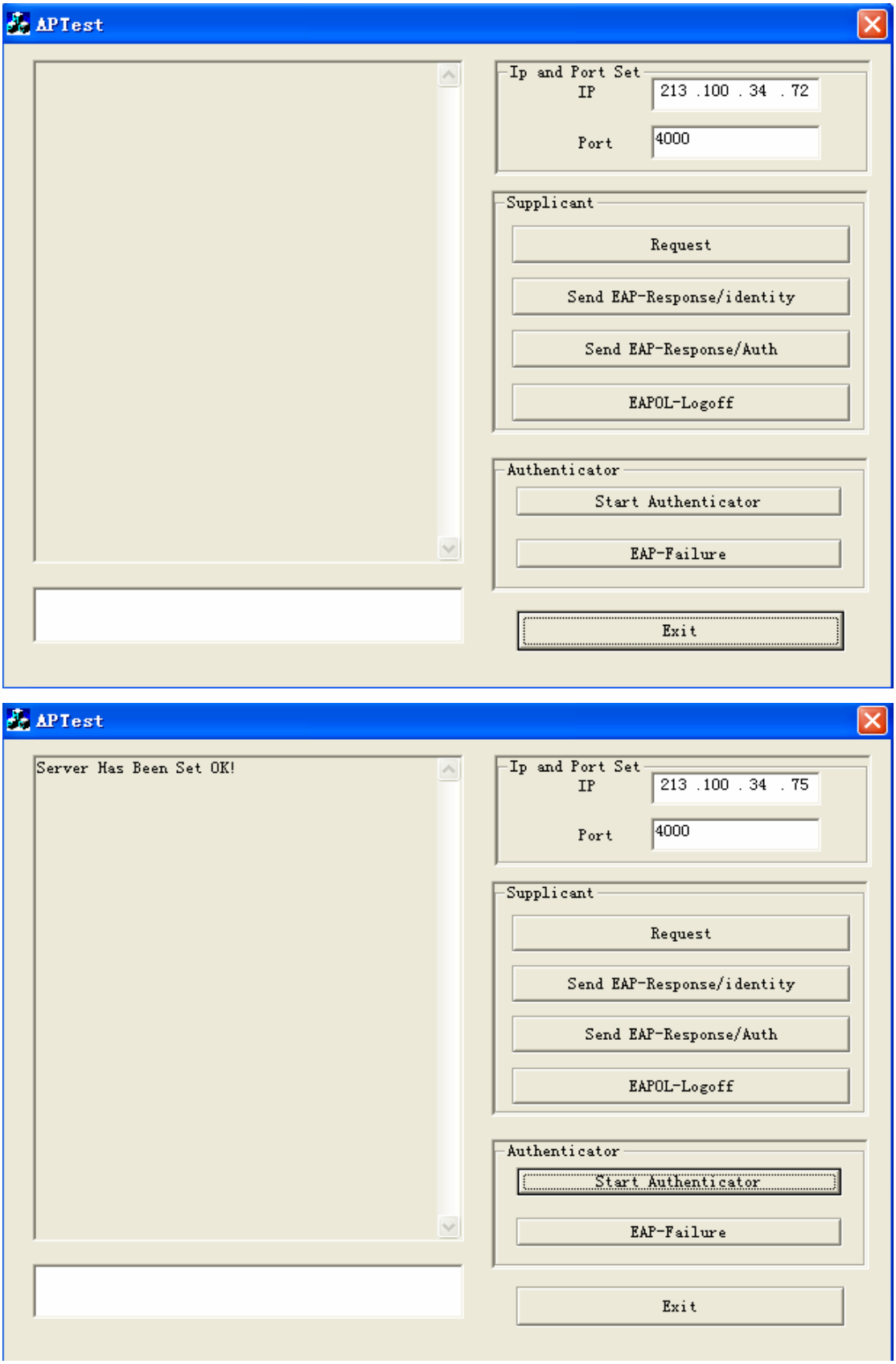


Figure 5.3 Starting the supplicant and authenticator programs

Before the supplicant clicks the “Request” button to send the EAP-Start, if the

user tries to access the network, a popup window (shown in Figure 5.4) will indicate that there was no connection between the supplicant and the authenticator. Thus initially the supplicant has no connectivity and no bandwidth available to them.

Figure 5.4 Before the supplicant sent EAP-Start

Next the user clicks on the supplicant's "Request" button. The interaction information is shown in the interaction information area (Figure 5.5) - this shows that the supplicant has set an EAPOL-Start message and in response the authenticator has sent an EAP Request / Identity message.

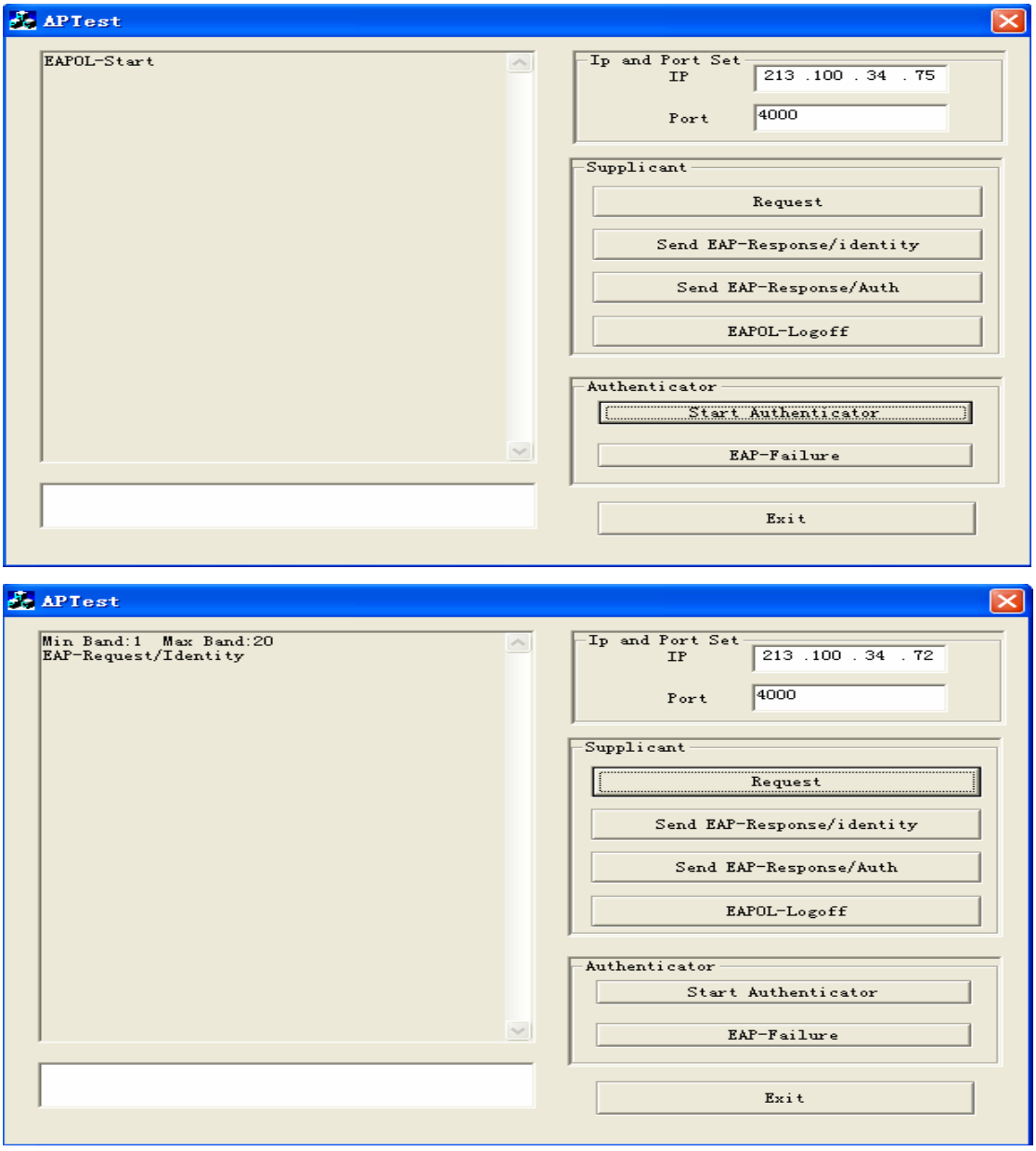


Figure 5.5 Interaction information shown after clicking Request (the upper figure shows the authenticator's interface and the lower figure is supplicant's)

As shown in Figure 5.5, the authenticator received an “EAP-Start” request from the supplicant and responded to this request with an “EAP-Request/Identity” and “Min Band 1 Max Band 20” message. This information shows that the authenticator requested the supplicant to send its identity and that it offered the supplicant an access bandwidth which was no more than 20 (in this case the units are KB per second). After learning this bandwidth value, the supplicant can access the network. We entered 5 characters and 25 characters separately (representing 5K/sec and 25K/sec separately) in the blank area to see the result. The test results are shown in Figure 5.6 and Figure 5.7.

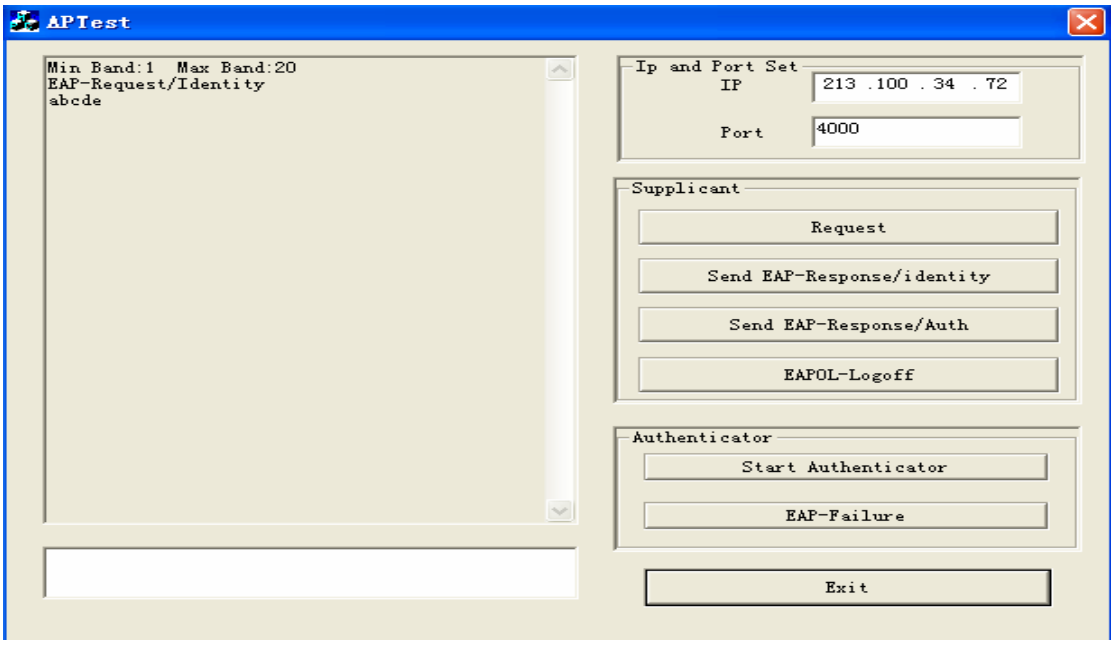


Figure 5.6 Enter 5 characters

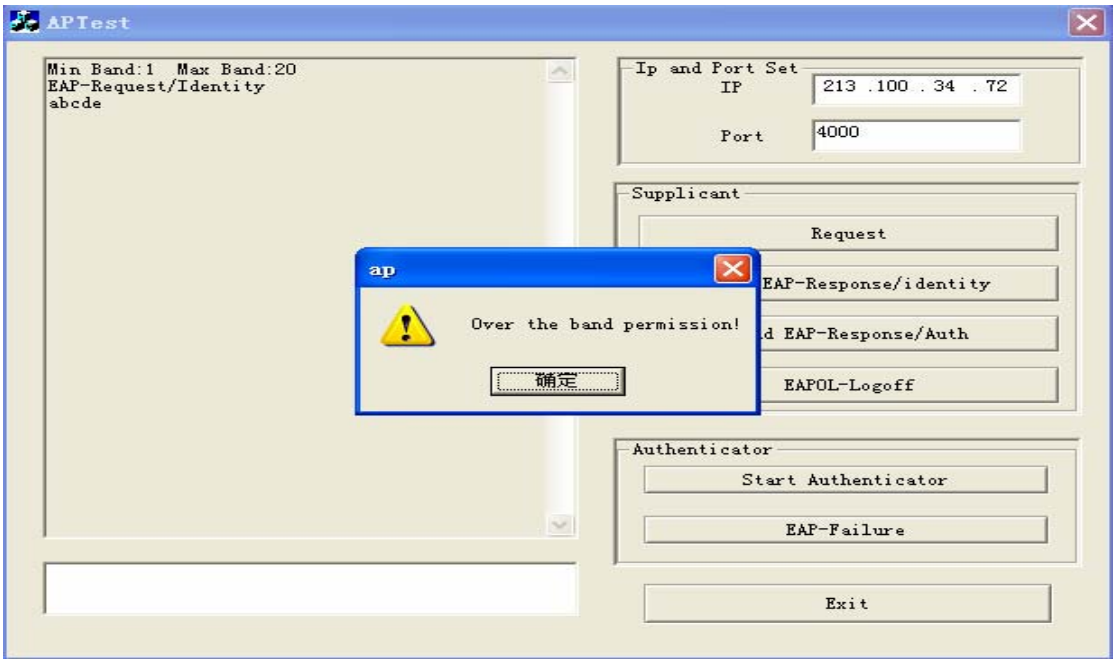


Figure 5.7 The user interface of the authenticator (top) and supplicant (bottom) after the supplicant has entered 25 characters

As shown in Figure 5.6 and Figure 5.7, when the number of characters entered were less than 20, then the characters could be shown in the interaction area of the supplicant and authenticator after we pressed “Enter” button on the laptop. Thus the supplicant could access the network although the available bandwidth was low. When the number of characters entered were more than 20, after we pressed “Enter” button, a popup window indicates that the packets that the supplicant attempted to send were over the permitted bandwidth. Therefore, the packets could not be sent to the destination and were discarded.

Following this, we clicked the supplicant’s “Send EAP-Response/Identity” button, and this was shown in the information interaction area. In the actual authentication process, the authenticator should forward this information to the authentication server. In our emulation process, we did not emulate the interaction between the authenticator and the authentication server. Details of this interaction can be found in Zhou Jia’s thesis[39]. Note that authenticator does not have to know which kind of authentication method was used. It simply forwards the authentication request from the authentication server to the supplicant; this information is shown in the information interaction area of the supplicant (as shown in Figure 5.8).

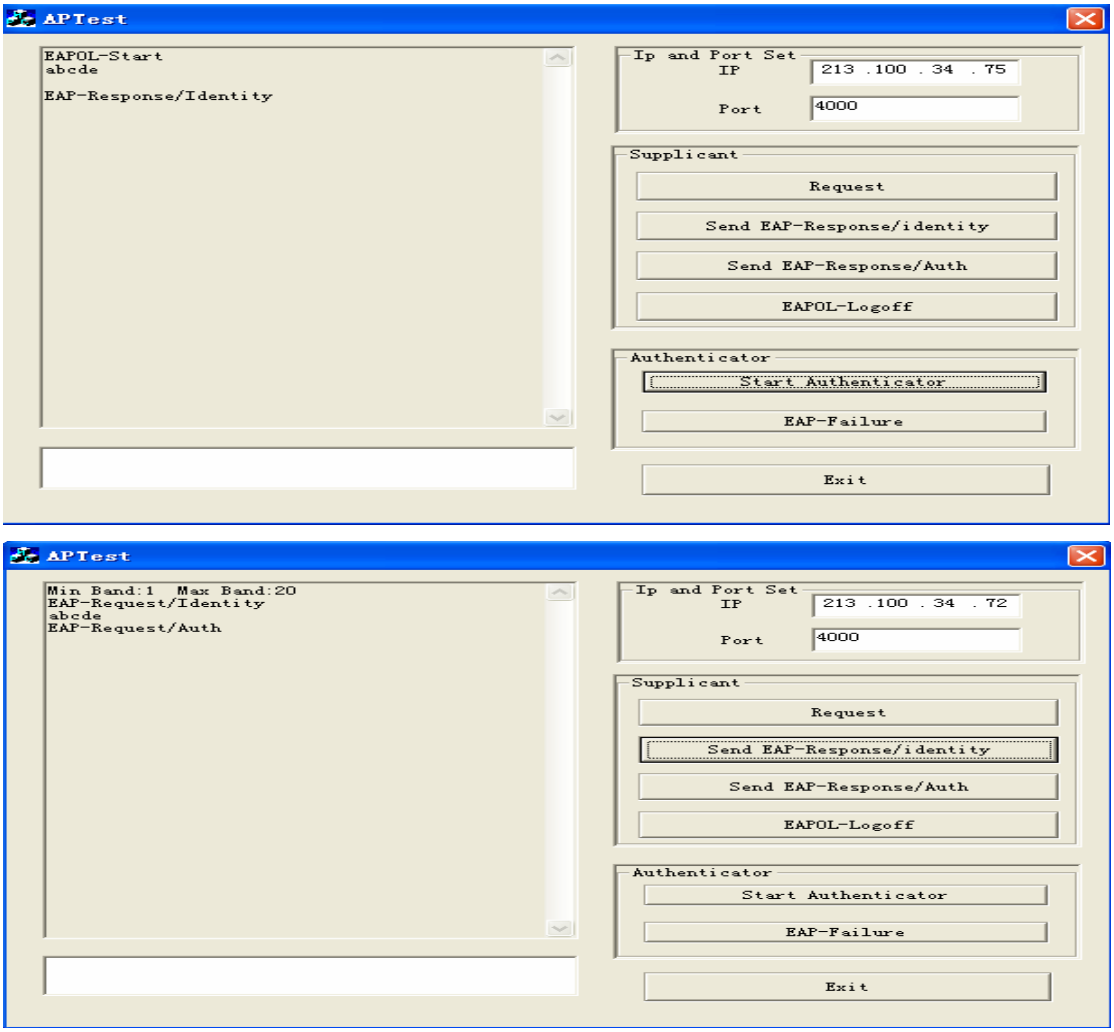


Figure 5.8 The second step of the information interaction

Next we clicked the supplicant's "Send EAP-Response/Auth" button. First we examine the case of an authentication success. In this case, in the supplicant's information interaction area, "EAP-Success/MAX Band:50" was shown, indicating that the supplicant's identity had been authenticated and the maximum permitted bandwidth to this supplicant was increased to 50. As the supplicant was allocated greater bandwidth we were able to enter more characters of text and successfully sent them. See Figure 5.9 and Figure 5.10.

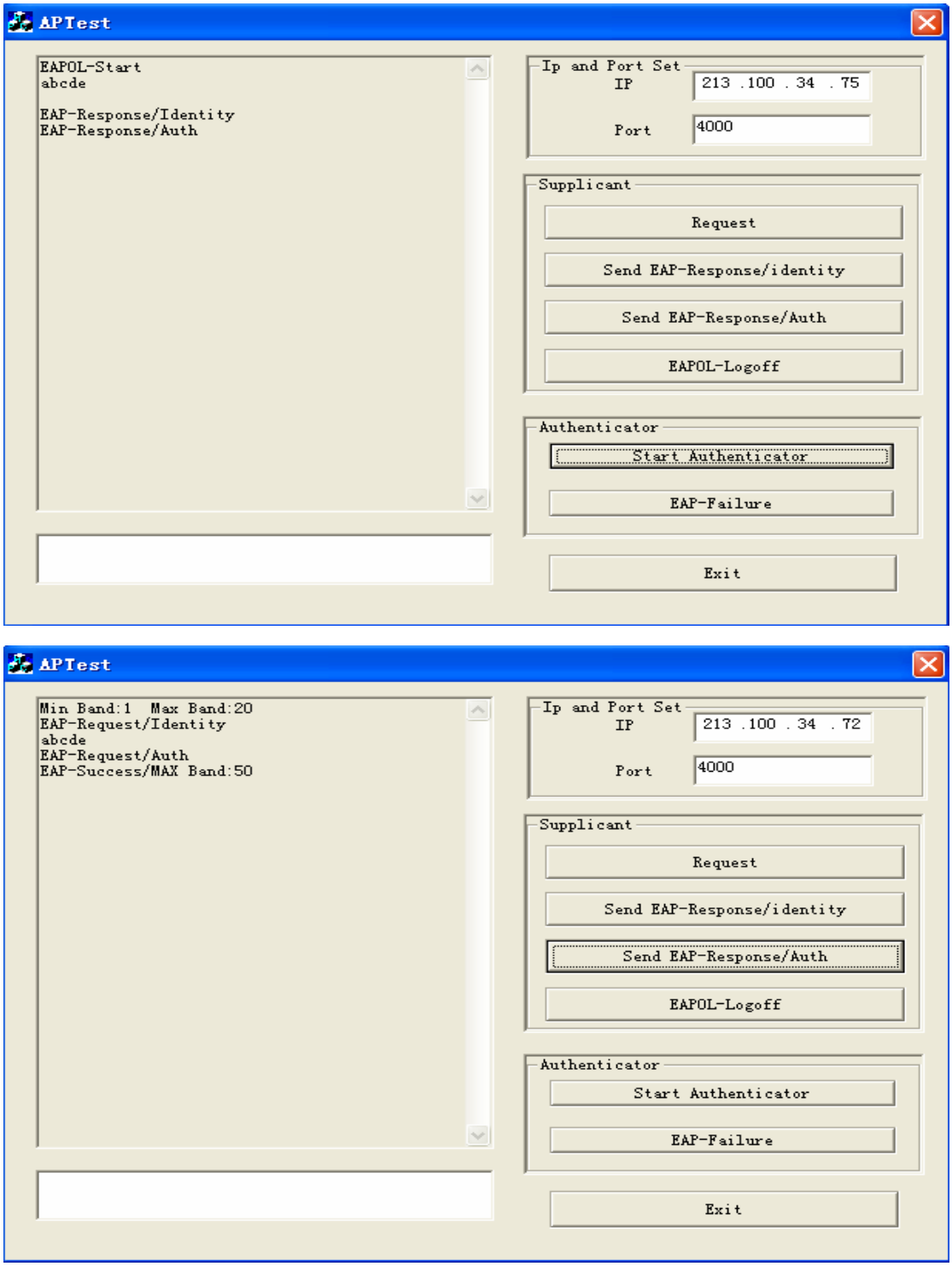


Figure 5.9 Authentication Success information

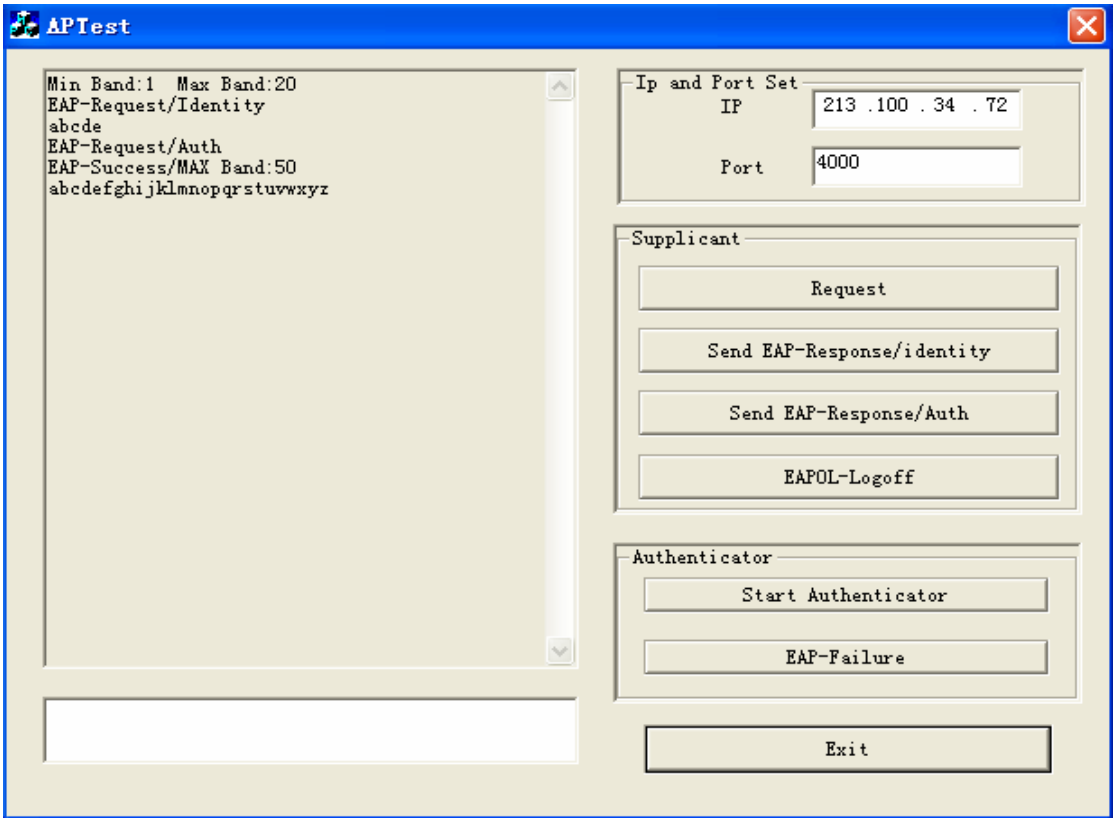


Figure 5.10 As the supplicant was assigned more bandwidth a longer text string could be successfully sent

In the case that the supplicant was not successfully authenticated, then the supplicant receives an EAP-Failure message. After this, we were unable to enter characters in the blank area since access was denied due to the authentication failure. See Figure 5.11 and Figure 5.12. The Wireshark capture of the traffic in the emulator is shown in Figure 5.13.

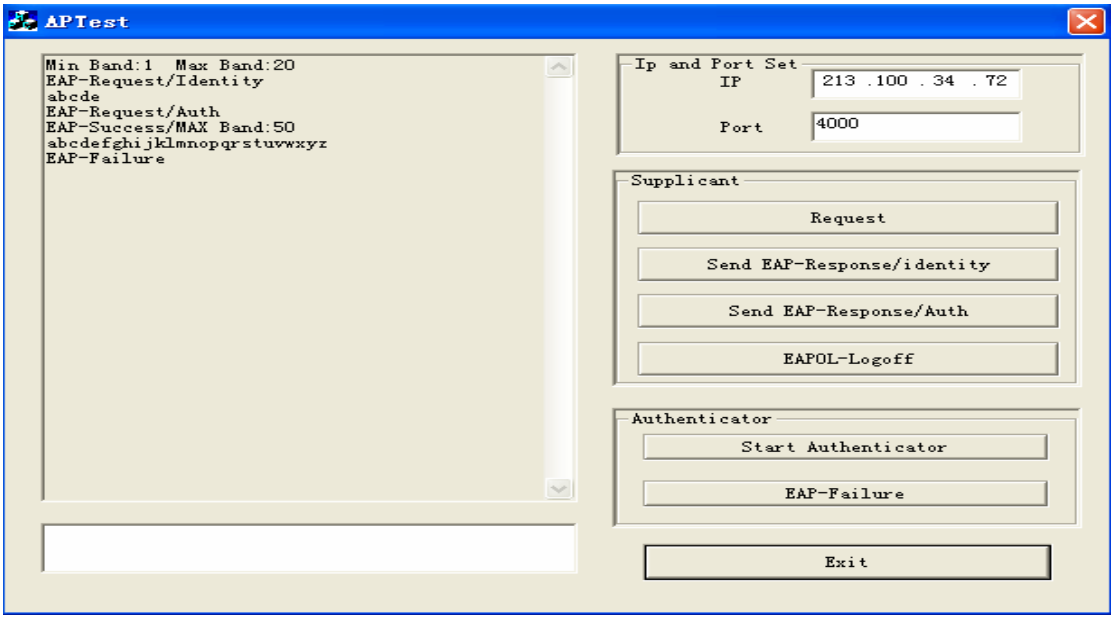


Figure 5.11 Authentication Failure information

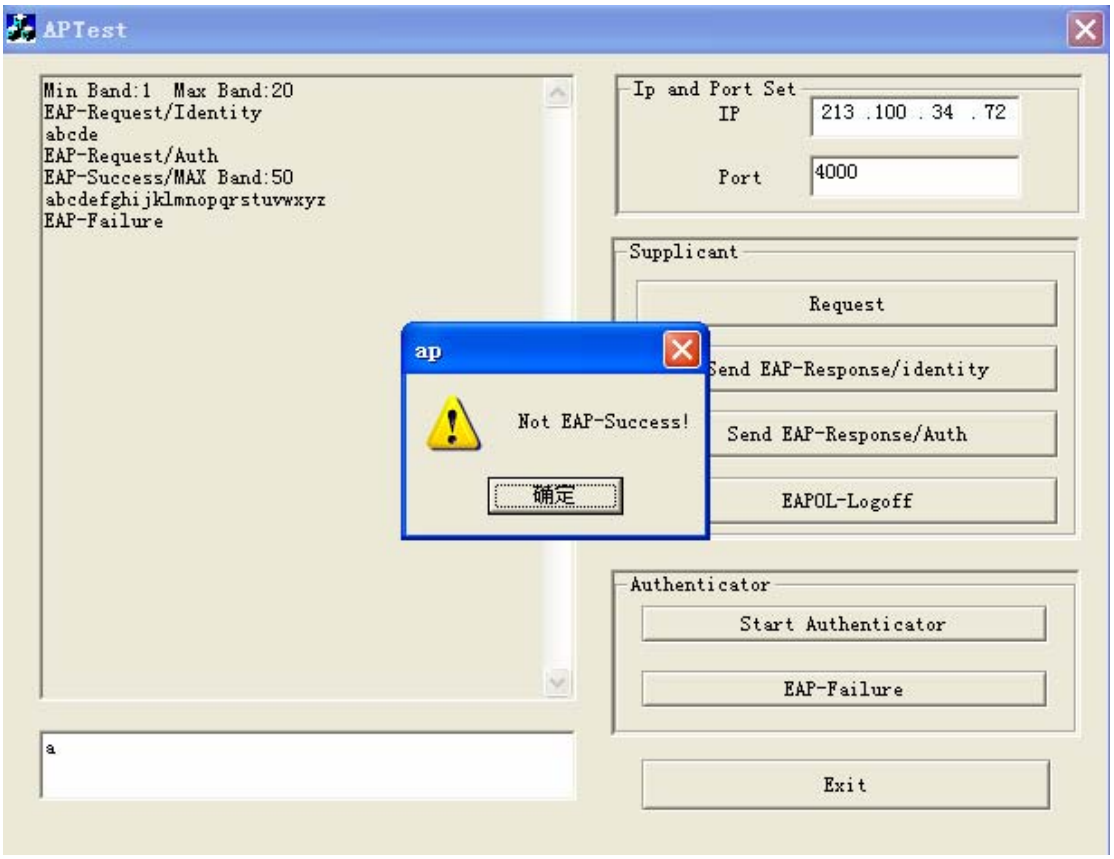


Figure 5.12 The supplicant's access was denied

Source	Destination	Protocol	Info
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [SYN] Seq=0 Win=16384 Len=0 MSS=1460
213.100.34.149	213.100.34.72	TCP	terabase > wysdmc [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=14
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [ACK] Seq=1 Ack=1 Win=17520 [TCP CHECKSUM INC
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [PSH, ACK] Seq=1 Ack=1 Win=17520 [TCP CHECKSU
213.100.34.149	213.100.34.72	TCP	terabase > wysdmc [PSH, ACK] Seq=1 Ack=14 Win=65522 Len=47
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [ACK] Seq=14 Ack=48 Win=17473 [TCP CHECKSUM I
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [PSH, ACK] Seq=14 Ack=48 Win=17473 [TCP CHECK
213.100.34.149	213.100.34.72	TCP	terabase > wysdmc [PSH, ACK] Seq=48 Ack=37 Win=65499 Len=18
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [ACK] Seq=37 Ack=66 Win=17455 [TCP CHECKSUM I
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [PSH, ACK] Seq=37 Ack=66 Win=17455 [TCP CHECK
213.100.34.149	213.100.34.72	TCP	terabase > wysdmc [PSH, ACK] Seq=66 Ack=56 Win=65480 Len=25
213.100.34.72	213.100.34.149	TCP	wysdmc > terabase [ACK] Seq=56 Ack=91 Win=17430 [TCP CHECKSUM I

Figure 5.13 Capture of the traffic in the emulator

From above test results, the goal of the non-binary authentication has been achieved. The authentication system can control the allowed bandwidth to the supplicant dynamically, thus improving the perceived performance for valid users, while minimizing the resources given to invalid users.

6. Conclusions and Future Work

In this chapter, we will summarize our conclusions for this thesis project and propose some suggestions for future work.

6.1 Conclusions

In this thesis project, we have examined the capability of a supplicant to send and receive packets before and after it is authenticated. Also, we have implemented a new supplicant program which could test and stress the authenticator by sending EAP and UDP packets together. Since the new non-binary authenticator has not yet been implemented, the supplicant was unable to be tested together with the authenticator and the authentication server. However, we have emulated the interaction between the supplicant and the non-binary authenticator. The test results suggest that non-binary authentication is both possible and useful. A supplicant can utilize a limited amount of bandwidth even without being authenticated when it moves into the coverage of a non-binary authentication AP. Therefore the supplicant can continue a VoIP session or other activities without a major gap in traffic. The proposed authentication system can also dynamically and flexibly control the bandwidth allocated to the supplicant.

Non-binary authentication is an interesting topic. However, it will take additional time and effort to realize. This thesis project represents only a tiny step but hopefully it offers a good start. In the next section, some suggestions are given for future work.

6.2 Future work

One potential aspect of future work would be to add a bandwidth negotiation mechanism between the supplicant and the authentication server. Thus each supplicant could negotiate its desired bandwidth with the authentication server. However, the interaction of such a bandwidth negotiation mechanism among supplicant, authenticator, and authentication server is somewhat complicated, but could easily be done based on the prior work present in this thesis and the two accompanying theses[39, 43].

In addition, as another example of non-binary authentication, a supplicant for the HP iPAQ (a type of personal digital assistant) that could authenticate to the KTH's WLAN (KTHOPEN-WPA) once, then power down its radio to save power. When it wants to communicate again it will simply power up and continue without the user needing to enter their key again. In this way, it is more convenient for a PDA supplicant to access the network, and the authentication mechanism is more secure

than that of KTHOPEN. The supplicant’s configuration could be based on the new non-binary authenticator’s implementation.

References

[1] Jun Lei, Xiaoming Fu, Dieter Hogrefe, and Jianrong Tan, “Comparative Studies on Authentication and Key Exchange Methods for 802.11 Wireless LAN”, available at: http://www.net.informatik.uni-goettingen.de/publications/1358/PDF, last visited: March 2008.

[2] Kwang-Hyun Baek, Sean W. Smith, and David Kotz, “A Survey of WPA and 802.11i RSN Authentication Protocols”, Technial Report TR2004-524, Dartmouth College Computer Science, November 2004.

[3] Institute of Electrical and Electronics Engineers, available at: http://en.wikipedia.org/wiki/IEEE, last visited: March 2008.

[4] IEEE 802 LAN/MAN Standards Committee, available at: http://www.ieee802.org/, last visited: March 2008.

[5] William A. Arbaugh, Narendar Shankar, Y. C. Justin, and Kan Zhang, “Your 802.11 Wireless Network Has No Clothes”, IEEE Wireless Communications, 9(6): pp. 44-51, December 2002.

[6] Wi-Fi Protected Access, available at: http://www.wi-fi.org/knowledge_center_overview.php?docid=4486, last visited: March 2008.

[7] Linux WPA/WPA2/IEEE 802.1X Supplicant, available at: http://hostap.epitest.fi/wpa_supplicant/, last visited: March 2008.

[8] W.Simpson, “The Point-to-Point Protocol (PPP)”, RFC 1661, July 1994, available at: http://www.ietf.org/rfc/rfc1661.txt?number=1661, last visited: March 2008.

[9] Point-to-Point Protocol, available at: http://en.wikipedia.org/wiki/Point-to-Point_Protocol, last visited: March 2008

[10] L. Blunk and J. Vollbrecht, “PPP Extensible Authentication Protocol (EAP)”, RFC 2284, March 1998, available at: http://www.ietf.org/rfc/rfc2284.txt?number=2284, last visited: March 2008.

[11] Lars Strand, “802.1X Port-Based Authentication HOWTO”, August 18th, 2004, available at: http://tldp.org/HOWTO/html_single/8021X-HOWTO/, last visited: March 2008.

[12] Yu-Ping Wang, Yi-Wen Liu, and Jyh-Cheng Chen, “Design and Implementation of WIRE1x”, National Tsing Hua University, Taiwan, available at: wire.cs.nthu.edu.tw/wire1x/TANET03.pdf, last visited: March 2008.

[13] B. Aboba and D. Simon, “PPP EAP TLS Authentication Protocol”, RFC 2716, October 1999, available at: http://www.ietf.org/rfc/rfc2716.txt?number=2716, last visited: March 2008.

[14] W. Simpson, “PPP Challenge Handshake Authentication Protocol (CHAP)”, RFC 1994, August 1996, available at: http://www.ietf.org/rfc/rfc1994.txt?number=1994, last visited: March 2008.

[15] Paul Funk, “EAP Tunneled TLS Authentication Protocol (EAP-TTLS)”, March 2002, available at: http://www.ietf.org/proceedings/02mar/slides/eap-1/, last visited: March 2008.

[16] H. Andersson, S. Josefsson, G. Zorn, and B. Aboba, “Protected Extensible Authentication Protocol (PEAP) <draft-josefsson-pppext-eap-tls-eap-01.txt >”, Internet Draft, October 2001, available at: http://ietfreport.isoc.org/all-ids/draft-josefsson-pppext-eap-tls-eap-01.txt, last visited: March 2008.

[17] B. Aboba, L. Blunk, J. Vollbrecht, and J. Carlson, “Extensible Authentication Protocol (EAP)”, RFC 3748, June 2004, available at: http://www.ietf.org/rfc/rfc3748.txt?number=3748, last visited: March 2008.

[18] IEEE, “IEEE Standards for Local and Metropolitan Area Networks: Standard for Port Based Network Access Control”, IEEE Std 802.1x-2004, October 2004, available at: http://standards.ieee.org/getieee802/download/802.1X-2004.pdf, last visited: March 2008.

[19] Joe Salowey, Hao Zhou, S. Josefsson, Glen Zorn, Dan Simon, and Ashwin Palekar, “Protected Extensible Authentication Protocol (PEAP) Version 2 <draft-josefsson-pppext-eap-tls-eap-10.txt >”, Internet Draft, October 2004, available at: http://www.potaroo.net/ietf/idref/draft-josefsson-pppext-eap-tls-eap/, last visited: May 2008.

[20] PEAP Protected Extensible Authentication Protocol, available at: http://www.arlinx.com/PEAP-Protected-Extensible-Authentication-Protocol.html, last visited: April 2008.

[21] C. Rigney, S. Willens, A. Rubens, W. Simpson, “Remote Authentication Dial In User Service (RADIUS)”, RFC 2865, June 2000, available at: http://www.ietf.org/rfc/rfc2865.txt?number=2865, last visited: April 2008.

[22] RADIUS Server AAA Authentication Authorization Accounting, available at: http://www.arlinx.com/RADIUS.html, last visited: April 2008.

[23] G. Zorn, "Microsoft Vendor-specific RADIUS Attributes", RFC 2548, March 1999, available at: http://www.ietf.org/rfc/rfc2548.txt?number=2548, last visited: April 2008.

[24] 802.1x Port Based Network Access Control, available at: http://wiki.personaltelco.net/index.cgi/PortBasedNetworkAccessControl, last visited: April 2008.

[25] OpenSEA Alliance Formed by Leading Vendors to Develop and Distribute Open Source 802.1X Supplicant, available at: http://www.openseaalliance.org/index.php?option=com_content&task=view&id=6&Itemid=40, last visited: April 2008.

[26] AEGIS (network), available at: http://en.wikipedia.org/wiki/AEGIS_(network), last visited: April 2008.

[27] Cisco Systems Completes Acquisition of Meetinghouse Data Communications, available at: http://newsroom.cisco.com/dlls/2006/corp_081606.html, last visited: April 2008.

[28] Cisco Secure Services Client Introduction, available at: http://www.cisco.com/en/US/products/ps7034/index.html, last visited: April 2008.

[29] Juniper Networks Odyssey Access Client, available at: http://www.juniper.net/products_and_services/aaa_and_802_1x/odyssey/odyssey_access_client/index.html, last visited: April 2008.

[30] Sean Michael Kerner, "Network Access Heats Up With 802.1x Funk", July 2006, available at: http://www.internetnews.com/security/article.php/3620336, last visited: April 2008.

[31] Juniper Networks Completes Acquisition of Funk Software, available at: http://www.juniper.net/company/presscenter/pr/2005/pr-051201.html, last visited: April 2008.

[32] IEEE 802.1x Open Source Implementation, available at: http://open1x.sourceforge.net/, last visited: April 2008.

[33] Linux WPA/WPA2/IEEE 802.1X Supplicant, available at: http://hostap.epitest.fi/wpa_supplicant/, last visited: April 2008.

[34] Wire1x, available at: http://wire.cs.nthu.edu.tw/wire1x/, last visited: April 2008.

[35] Linux WPA/WPA2/IEEE 802.1X Supplicant, available at: http://hostap.epitest.fi/wpa_supplicant/, last visited: April 2008.

[36] About Wireshark, available at: http://www.wireshark.org/about.html, last visited: July 2008.

[37] Wireshark, available at: http://en.wikipedia.org/wiki/Wireshark, last visited: July 2008.

[38] Wireless network, available at: https://dsv.su.se/en/comp/wireless, last visited: August 2008.

[39] Zhou Jia, "Adding bandwidth specification to a AAA Server", Master thesis, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, September, 2008.

[40] Alert Standard Format (ASF) Specification, available at: http://www.dmtf.org/standards/asf/, last visited: November 2008.

[41] Thomas Otto, "Extensible Network Access Authentication", Diplomarbeit (Diplom thesis), TU Braunschweig, July 4, 2006, available at: www-public.tu-bs.de:8080/~y0013790/thesis-otto-eapmethods.pdf, last visited: November 2008.

[42] Transport Layer Security, available at: http://en.wikipedia.org/wiki/Transport_Layer_Security, last visited: November 2008.

[43] Guo Jia, "Non-binary authentication", Master thesis, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, 2008 (draft)

[44] C. Rigney, A. Rubens, W. Simpson, S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2058, January 1997, available at: http://tools.ietf.org/html/rfc2058 , last visited: December 2008.

Appendix A Source code for the new supplicant

```
//////////////////////////////added by hengchong ////////////////////////////////////
typedef struct {
    u_char DesMAC[6];
    u_char SrcMAC[6];
    USHORT Ethertype;
    }DLC_Header;
typedef struct {
    u_char h_len;
    u_char tos;
    USHORT total_len;
    USHORT ident;
    USHORT frag_and_flags;
    u_char ttl;
    u_char proto;
    USHORT checksum;
    u_char sourceIP[4];
    u_char destIP[4];
    }IpHeader;
typedef struct _UDP{
    USHORT SrcPort;
    USHORT DstPort;
    USHORT Len;
    USHORT Chksum;
    }UDP_Header;
typedef struct dns_header
{
    USHORT id;
    USHORT flags;
    USHORT questions;
    USHORT answers;
    USHORT author;
    USHORT addition;
}DNS_HEADER;
typedef struct tsd_hdr
{
    BYTE saddr[4];
    BYTE daddr[4];
    BYTE mbz;
    BYTE ptcl;
    USHORT udpl;
}PSD_HEADER;
```

```
/*********************************************************************************************
*************
*    Name: //Calculate Checksum
*    Created: hengchong
*********************************************************************************************
*************/
USHORT checksum(USHORT *buffer,int size)
{
     unsigned long cksum=0;
     while(size>1)
     {
          cksum+=*buffer++;
          size-=sizeof(USHORT);
     }
     if(size)
     {
          cksum+=*(UCHAR *)buffer;
     }
     while (cksum>>16)
          cksum=(cksum>>16)+(cksum & 0xffff);
     return (USHORT) (~cksum);
}
/*********************************************************************************************
*    Name: Make udp Package
*    Created: Hengchong
*********************************************************************************************/
int MakeUdp(u_char * udpPackage)
{
     //Construct a domain to store packets
//   u_char *udppackage;
     u_char *src_addr;
     u_char domain[50];//domain name
     DLC_Header *dlcheader,*fakedlc;
     IpHeader *ipheader,*fakeip;//IP header
     UDP_Header *fakeudp;//UDP header
     DNS_HEADER *fakedns;//DNS header
     PSD_HEADER *psdheader;
//   USHORT ip_len;//IP header length
     char srcip[4];//Source IP address
     char destip[4];//Destination IP address
     char srcmac[6];//Source MAC address
//   char destmac[6];// Destination MAC address
//   char domain[50];
```

```
    //Fake IP address infomation
    memset(srcip,0,4);
    memset(destip,0,4);
    //Construct fake answering packets
        fakedns=(DNS_HEADER *)malloc(12+strlen((char *)domain)+21);//Assign memory for DNS
packets
        memset(fakedns,0,12+strlen((char *)domain)+21);
        fakedns->id=2222;
        fakedns->flags=htons(0x8180);
        fakedns->questions=htons(1);
        fakedns->answers=htons(1);
        fakedns->author=0;
        fakedns->addition=0;
        memcpy((char *)fakedns+12,domain,strlen((char *)domain)+1);
        *((BYTE *)fakedns+12+strlen((char *)domain)+1)=0x00;//Type query
        *((BYTE *)fakedns+12+strlen((char *)domain)+2)=0x01;
        *((BYTE *)fakedns+12+strlen((char *)domain)+3)=0x00;
        *((BYTE *)fakedns+12+strlen((char *)domain)+4)=0x01;
        *((BYTE *)fakedns+12+strlen((char *)domain)+5)=0xC0;
        *((BYTE *)fakedns+12+strlen((char *)domain)+6)=0x0C;
        *((BYTE *)fakedns+12+strlen((char *)domain)+7)=0x00;
        *((BYTE *)fakedns+12+strlen((char *)domain)+8)=0x01;
        *((BYTE *)fakedns+12+strlen((char *)domain)+9)=0x00;
        *((BYTE *)fakedns+12+strlen((char *)domain)+10)=0x01;
        *((BYTE *)fakedns+12+strlen((char *)domain)+11)=0x00;//TTL
        *((BYTE *)fakedns+12+strlen((char *)domain)+12)=0x00;
        *((BYTE *)fakedns+12+strlen((char *)domain)+13)=0x00;
        *((BYTE *)fakedns+12+strlen((char *)domain)+14)=0x80;
        *((BYTE *)fakedns+12+strlen((char *)domain)+15)=0x00;//IP address length
        *((BYTE *)fakedns+12+strlen((char *)domain)+16)=0x04;
        //Fill fake IP address
        *((u_char *)fakedns+12+strlen((char *)domain)+17)=192;
        *((u_char *)fakedns+12+strlen((char *)domain)+18)=168;
        *((u_char *)fakedns+12+strlen((char *)domain)+19)=0;
        *((u_char *)fakedns+12+strlen((char *)domain)+20)=88;
        //Fill udp packets
        fakeudp=(UDP_Header *)malloc(8+12+strlen((char *)domain)+21);
        memset((char *)fakeudp,0,8+12+strlen((char *)domain)+21);
        fakeudp->DstPort=5555;
        fakeudp->SrcPort=6666;
        fakeudp->Chksum=0;//Check sum
        fakeudp->Len=htons(8+12+strlen((char *)domain)+21);//UDP packets size
        memcpy((char *)fakeudp+8,(char *)fakedns,12+strlen((char *)domain)+21);
        psdheader=(PSD_HEADER *)malloc(12+8+12+strlen((char *)domain)+21);
```

```
        psdheader->mbz=0;
        psdheader->ptcl=17;
        psdheader->udpl=htons(8+12+strlen((char *)domain)+21);
        memcpy((char *)&(psdheader->saddr[0]),destip,4);
        memcpy((char *)&(psdheader->daddr[0]),srcip,4);
        memcpy((char *)psdheader+12,(char *)fakeudp,8+12+strlen((char *)domain)+21);
        fakeudp->Chksum=checksum((USHORT *)psdheader,12+8+12+strlen((char *)domain)+21);
        //Fill IP packets
        fakeip=(IpHeader *)malloc(20+8+12+strlen((char *)domain)+21);
        memset((char *)fakeip,0,20+8+12+strlen((char *)domain)+21);
        fakeip->h_len=0x45;
        fakeip->checksum=0;
        fakeip->tos=ipheader->tos;
        fakeip->total_len=htons(20+8+12+strlen((char *)domain)+21);
        fakeip->ident=ipheader->ident;//Identity
        fakeip->frag_and_flags=ipheader->frag_and_flags;
        fakeip->proto=ipheader->proto;
        fakeip->ttl=128;
        memcpy((char *)fakeip+12,(char *)destip,4);//Source IP address
        memcpy((char *)fakeip+16,(char *)srcip,4);//Destination IP address
        memcpy((char *)fakeip+20,(char *)fakeudp,8+12+strlen((char *)domain)+21);
        //Calculate Checksum
        fakeip->checksum=checksum((USHORT *)fakeip,20);
        //Fill MAC frame
        fakedlc=(DLC_Header *)malloc(14+20+8+12+strlen((char *)domain)+21);
        memset(fakedlc,0,14+20+8+12+strlen((char *)domain)+21);
       src_addr = get_src_mac();
        memcpy((char *)&(fakedlc->DesMAC[0]),eapol_dst,6);//Fill destination MAC
        memcpy((char *)&(fakedlc->SrcMAC[0]),src_addr,6);//Fill source MAC
        free(src_addr);
        src_addr = NULL;
        fakedlc->Ethertype=dlcheader->Ethertype;
        memcpy((char *)fakedlc+14,(char *)fakeip,20+8+12+strlen((char *)domain)+21);
        udpPackage=(u_char *)fakedlc;
        return (14+20+8+12+strlen((char *)domain)+21);
}
/*********************************************************************************************
*    Name: SendUdpPackage
*    Created:Hengchong
*********************************************************************************************/
void txSendUdp()
{
    printf("send txStart\n");
    u_char *temp=NULL;
```

```
    int len = MakeUdp(temp);
    if (send_frame(temp, len) == NULL)
        printf("send udp to link.\n");
    free(temp);
    temp = NULL;
}
```

Appendix B　Source code for emulation

```
//////////////////////////////////////////////////////////////////////////
// CClientSocket
CClientSocket::CClientSocket()
{
     m_aSessionIn=NULL;
     m_aSessionOut=NULL;
     m_sfSocketFile=NULL;
     m_bInit=false;
     m_bClose=false;
}
CClientSocket::~CClientSocket()
{
     if(m_aSessionIn)
          delete m_aSessionIn;
     if(m_aSessionOut)
          delete m_aSessionOut;
     if(m_sfSocketFile)
          delete m_sfSocketFile;
}
// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CClientSocket, CSocket)
     //{{AFX_MSG_MAP(CClientSocket)
     //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif    // 0
//////////////////////////////////////////////////////////////////////////
// CClientSocket member functions
void CClientSocket::OnReceive(int nErrorCode)
{
     // TODO: Add your specialized code here and/or call the base class
     CSocket::OnReceive(nErrorCode);
     do
     {
          CMessg temp;
          temp.Serialize(*m_aSessionIn);
          m_dlg->m_sMsgList+=temp.m_strText+"\r\n";
//MessageBox(NULL,temp.m_strText,"",MB_OK);
          if(!m_dlg->m_bClient)
          {
               for(POSITION pos=m_dlg->m_connectionList.GetHeadPosition();pos!=NULL;)
```

```
            {
                CClientSocket * t = (CClientSocket*)m_dlg->m_connectionList.GetNext(pos);
                //if(t->m_hSocket!=this->m_hSocket)
                {
//MessageBox(NULL,temp.m_strText,"",MB_OK);
                    if(temp.m_strText.Compare("EAPOL-Start")==0)
                    {
                        CMessg temp1;
                        temp1.m_strText ="Min Band:1  Max
Band:20\r\nEAP-Request/Identity";
                    //   m_dlg->bSuccessLittleBand=true;

                       t->SendMessage(&temp1);
                    //   m_dlg->m_sMsgList+="\n"+temp1.m_strText+"\n";
                    }else
                    if(temp.m_strText.Compare("EAP-Response/Identity")==0)
                    {
                        CMessg temp1;
                        temp1.m_strText ="EAP-Request/Auth";
                        t->SendMessage(&temp1);
                    //   m_dlg->m_sMsgList+="\n"+temp1.m_strText+"\n";
                    }else if(temp.m_strText.Compare("EAP-Response/Auth")==0)
                    {
                        CMessg temp1;
                        temp1.m_strText ="EAP-Success/MAX Band:50";
                        t->SendMessage(&temp1);
                        m_dlg->bSuccess=true;
                    }else if(temp.m_strText.Compare("EAPOL-Logoff")==0)
                    {
                        CMessg temp1;
                        temp1.m_strText ="Finish Logoff";
                        t->SendMessage(&temp1);
                        m_dlg->bSuccess=false;
                    }
                }
            }
        }else if(temp.m_strText.Compare("EAP-Success/MAX Band:50")==0)
        {
                m_dlg->bSuccess=true;
        }else if(temp.m_strText.Compare("Finish Logoff")==0)
        {
                m_dlg->bSuccess=false;
        }else if(temp.m_strText.Compare("EAP-Failure")==0)
        {
```

```
                    m_dlg->bSuccess=false;
                    m_dlg->bSuccessLittleBand=false;
          }else if(temp.m_strText.Compare("Min Band:1  Max
Band:20\r\nEAP-Request/Identity")==0)
          {
                         m_dlg->bSuccessLittleBand=true;
          }
          m_dlg->SetDlgItemText(IDC_SHOWTEXT,m_dlg->m_sMsgList);
          if(temp.m_tag==1&&m_dlg->m_willchating==FALSE)
          {
          //   memcpy(m_dlg->m_sound.m_cBufferOut,temp.m_buffer,MAX_BUFFER_SIZE);
          }
          int linenum=((CEdit *)(m_dlg->GetDlgItem(IDC_SHOWTEXT)))->GetLineCount();
          ((CEdit *)(m_dlg->GetDlgItem(IDC_SHOWTEXT)))->LineScroll(linenum);
          if(!m_dlg->m_bClient)
          {
               for(POSITION pos=m_dlg->m_connectionList.GetHeadPosition();pos!=NULL;)
               {
                    CClientSocket * t = (CClientSocket*)m_dlg->m_connectionList.GetNext(pos);
                    if(t->m_hSocket!=this->m_hSocket)
                    {
                         t->SendMessage(&temp);
                    }
               }
          }
     }
     while (!m_aSessionIn->IsBufferEmpty());
}

CServerSocket::CServerSocket()
{
}
// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CServerSocket, CSocket)
     //{{AFX_MSG_MAP(CServerSocket)
     //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif    // 0
//////////////////////////////////////////////////////////////////////////
// CServerSocket member functions
BOOL CServerSocket::Init(UINT port)
{
     m_uPort=port;
```

```
    m_dlg=dlg;
    if(Create(m_uPort)==FALSE)
    {
        AfxMessageBox("Server Socket Create Error");
        return FALSE;
    }
    if(this->Listen()==FALSE)
    {
        AfxMessageBox("Server Listen Error");
        return FALSE;
    }
    m_dlg->SetDlgItemText(IDC_SHOWTEXT,"Server Has Been Set OK!");
    return TRUE;
}
void CServerSocket::OnAccept(int nErrorCode)
{
    // TODO: Add your specialized code here and/or call the base class
    m_dlg->ProcessPendingAccept();
    CSocket::OnAccept(nErrorCode);
}
```

COS/CCS 2009-01

www.kth.se