Design and Implementation of WIRE Diameter

by

Wen-Ting Wu

July 2004

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE INSTITUE OF COMMUNICATION ENGINEERING

OF THE NATIONAL TSING HUA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER

To my parents

Acknowledgments

I would like to express sincere appreciation to my advisor, Professor Jyh-Cheng Chen of National Tsing Hua University, for the assistance of this research. His invaluable experience and advise make this thesis complete. I also acknowledge members of WIRE Lab for their kind support and encouragement. My gratitude must thank those people who helped me in those days, especially Ares Yeh. He is my boyfriend and sincerest partner.

Abstract

Design and Implementation of WIRE Diameter

Wen-Ting Wu

Advisor: Dr. Jyh-Cheng Chen

Although current network entities have increased in complexity, the issue of security is more important day by day. The AAA (Authentication, Authorization, and Accounting) protocol provides integrated user access control machination, so the Internet Engineering Task Force (IETF) formed a working group for discussing and formulating relative specifications. In some researches and experiments, traditional RADIUS protocol already is validated that it have much security weaknesses. Diameter thus was proposed to extend the RADIUS protocol and strengthen the security. Many actual application also agree that Diameter will replace RADIUS, for instance 3GPP. Then Diameter is regard as respectably expectable AAA protocol. Now, OpenDiameter[1] only bring up partial program library about Diameter protocol, but it is not any organization to finish whole Diameter testbed.

This thesis is design and implementation of WIRE Diameter, it emphasizes Authentication and Authorization. The WIRE Diameter is an open source, developed by the Wireless Internet Research & Engineering (WIRE) Laboratory, and sponsored this project by Industrial Technology Research Institute, Taiwan (ITRI [2]). This is an AAA server which could authenticates and authorize any PPP supplicant. The WIRE Diameter not only is based on

Diameter Base Protocol[3], but follows Diameter EAP Application[4]. The WIRE Diameter provides some authentication methods such as EAP-MD5, EAP-TLS, EAP-TTLS and EAP-PEAP. In this implementation, we also complete the NAS (Network Access Server) for overcoming AP (Access Point) without supporting Diameter. In this thesis, it is fully demonstrate how to design and implement the WIRE Diameter[5].

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In modern network environments, we have to use some authentication methods to gain network access. The device which provides accessibility must have a set of processes and protocols to verify user's qualification. Is the user indeed who he claims to be ? Is this user allowed to use services requested for ? Obviously, there is a need to focus on monitoring usage in heterogeneous networks. Internet Service Provider (ISP) must offer services other than standard dial-up, include of ISDN, xDSL, and cable-modem connectivity, even wireless situation. There is a need of standard way for verifying user's logon , monitoring user's network usage and billing user. A set of standards and protocols that can fulfill this agenda was named Authentication, Authorization and Accounting (AAA) protocol. AAA protocols mentioned today consists of two widely discussed protocol which is Remote Authentication Dial In User Service (RADIUS) [6] and Diameter [3].

The AAA Working Group [7] was formed by the Internet Engineering Task Force (IETF) to create a functional architecture that address the AAA model. This model focuses on the three crucial aspects of user access control: Authentication, Authorization, and Accounting respectively. The first RADIUS specification was proposed to be a standard in 1997. Many articles extends its ability and commercial products adopts RADIUS to provide AAA model. However, network entities have increased in complexity, deficiencies in the

RADIUS have been identified. In some researches and experiments, RADIUS protocol is already validated that it have many security weaknesses. Diameter thus was proposed to extend the RADIUS protocol and strengthen the security. Many actual applications also agree that Diameter will replace RADIUS, for instance 3GPP. The 3GPP specification [8] [9] notifies that some interfaces must based on Diameter for transport protocol. Then Diameter is regard as respectably expectable AAA protocol. There is a project to implement Diameter protocol in Open Community which is named as OpenDiameter. It's aimed at implementing functionality conformance to Diameter protocol specification. At current, OpenDiameter [1] only bring up partial program library about Diameter protocol, but there is not any organization to finish whole Diameter testbed.

This thesis presents the design and implementation of WIRE Diameter[5]. The WIRE Diameter is developed by the Wireless Internet Research & Engineering (WIRE) Laboratory based on OpenDiameter library. This is an AAA server which could authenticate and authorize any 802.1x[10] supplicant. The WIRE Diameter is not only based on Diameter Base Protocol[3], but follows Diameter EAP Application[4] specification. In current, the WIRE Diameter provides authentication methods such as EAP-MD5, EAP-TLS, EAP-TTLS and EAP-PEAP. In this implementation, we also complete the Network Access Server(NAS) for overcoming Access Point(AP) which is not supporting Diameter yet. In this thesis, it is fully demonstrated how to design and implement the WIRE Diameter.

1.1 Diameter Protocol

The Diameter base protocol is developed by IETF to provide an AAA framework for various applications such as network access, IP mobility[11] or EAP[4]. Diameter is also intended to work in local authentication, authorization, accounting and roaming situations. The Diameter base application needs to be supported by all Diameter implementations. The criteria for AAA protocol are summarized in [12]. It includes Failover, Transmission-level security,

Reliable transport, Agent support, Server-initiated message, Auditability, Transition support, Capability negotiation, Peer discovery and configuration and Roaming support.

In the following sections, we first compare between RADIUS and Diameter protocol to see benefits of Diameter protocol. Then, an overview of the Diameter base protocol and Diameter EAP application is presented.

1.1.1 Comparison between RADIUS and Diameter Protocol

RADIUS[6] has been developed and published by IETF to provide the AAA service. There are various implementations for different operation systems that have been demonstrated, e.g FreeRadius[13], CistronRadius, and GNU-radius. The FreeRadius have become widely used in the free-source community currently. RADIUS is considered to be insufficient to against some threats. We list below major issues about RADIUS and Diameter in related analysis, such as [3] and [14].

- **Failover Mechanism**

  Transport Failure is detected by peer. It is necessary for all pending request messages to be forwarded to an alternate agent if possible. This is commonly referred to Failover mechanism. Then Failover is significant for security authorities, it must ensure the service provisioning to be uninterruptible.

  RADIUS

  RADIUS is not support Failover, but leave it to implementations.

  Diameter

  For Failover mechanism, Diameter supports application-layer acknowledgements, and defines Failover algorithms and the associated state machine. This is described in Section 5.5 (Transport Failure Detection) of [3] and AAA-Transport-Profile[15]. Peers are able to do device watchdog behavior and communicate with DWR/DWA (These expressions will be defined in the section 2.3 in this thesis), for immediately detecting transport failure.

- **Transmission-Level Security (Authentication and Integrity)**

  End-to-end security services can be provided by supporting message integrity and confidentiality between two peers, even communicating through agent.

  RADIUS

  In typical RADIUS, it does not support the per-packet confidentiality. Instead, it defines application-layer authentication and integrity scheme that is required only for Response packets. [16] uses IP security (IPSec [17])on RADIUS and adopt IPSec by the option.

  Diameter

  Diameter provide universal support for transmission-level security, and enable both intra- and inter-domain AAA deployments. IPSec support is mandatory and TLS support is optional for NAS. [3] also claims that all Diameter implementations must support IPSec ESP [18] in transport mode with non-null encryption and authentication algorithms to provide per-packet authentication, integrity protection and confidentiality, and must support the replay protection mechanisms of IPSec.

- **Reliable Transport**

  To decrease packet loss is important principle for accounting and particular services. Loss packet may translate directly into revenue loss.

  RADIUS

  RADIUS runs over UDP, typical RADIUS is absence of retransmission behavior. Therefore, [16] consider the retransmission. Implementations must take responsible for maintain its own reliable transportation in its running environment.

  Diameter

  Diameter is specified to run over the reliable TCP or Stream Control Transmission Protocol (SCTP [19]) protocol.

- **Accounting Support**

  This is a major issue for the accounting services as described in [20]. Because lost

accounting packets may lose the revenue of service providers. Several fault resilience methods have been built into the accounting protocol in order minimize loss of accounting data in various fault situations under different capabilities assumptions of used devices.

RADIUS

There are some limitations in reliability and Failover mechanisms within RADIUS. Accounting of RADIUS [21] is an event-based accounting mechanism, that is to say it only support batch accounting.

Diameter

The accounting protocol of Diameter is based on a server directed model with capabilities for real-time delivery of accounting information. So Diameter provide interim and real-time accounting.

- **Agent Support**

  RADIUS

  RADIUS does not explicitly define any agent functionality.

  Diameter

  Diameter support agent behaviors including *relay*, *proxy*, *redirect* and *translation* agent, while providing agent aspects can simplify the network architectures. A Diameter request messages may be sent to a Diameter agent. The relay agent routes the request messages to the destined Diameter server by looking up its routing table. This routing table is called Realm-based Routing Table. It contains the route information of every realm. The proxy agent relays the request messages similar to that of a relay agent. It however can modify the messages to implement policy enforcement. A proxy agent should also maintain the state of the sending peer to provide admission control. When a request is received by a redirect agent, it replies an answer to instruct the sending peer where to forward this request. The translation agent provides the translation between two protocols. When a translation agent receives a request message in RADIUS format, it translates the request to a Diameter message.

- **Server-Initiated Messages**

  Server-initiated messages contain features such as unsolicited disconnect or re-authentication / re-authorization on demand across a heterogeneous deployment.

  RADIUS

  RADIUS server-initiated messages are defined in [22] as a option. This may encumber the implementation of some features.

  Diameter

  For improving RADIUS, Diameter provide server-initiated messages as a mandatory capability.

- **Audit-ability**

  The Audit-ability property allows the system to detect un-trusted proxies modifying attributes or even packet headers.

  RADIUS

  Data-object security mechanisms are not supported within RADIUS. A un-trusted RADIUS server may modify the data attributes and packet header during the proxy and this behavior may not be detected by the originators.

  Diameter

  In order to prevent from unauthorized access, Diameter defines the data object security and capabilities negotiation in [23], which is not mandatory.

- **Capability Negotiation**

  Capability negotiation allows the discovery of peer capability, protocol version, supported applications, security mechanisms, etc.

  RADIUS

  This mechanism is not supported within RADIUS.

  Diameter

  However Diameter support error handling, capability negotiation and mandatory / non-

mandatory AVPs in base protocol. When Diameter peers establish a transport connection, they must exchange the Capabilities Exchange messages, as specified in the peer state machine. A Diameter node must cache the supported applications in order to ensure that unrecognized commands and AVPs are not unnecessarily transmitted to peer.

- **Peer Discovery and Configuration**

  RADIUS needs manual configuration for peers. RADIUS implementations typically require that the name or address of servers or clients be manual configured, along with the corresponding shared secrets. The *Diameter* through Service Location Protocol (SLP) / Domain Name Server (DNS), Diameter enables dynamic discovery of peers. Derivation of dynamic session keys is enabled via transmission-level security.

- **Roaming Support**

  RADIUS

  Since RADIUS does not provide explicit support for proxies, and lacks audit-ability and transmission-level security features. RADIUS-based roaming is vulnerable to attack from external parties as well as susceptible for fraud perpetrated by the roaming partners themselves. As a result, RADIUS is not suitable for global roaming in open environments due to lack of security.

  Diameter

  Diameter-based roaming is secure and scalable. By providing explicit support for inter-domain roaming and message routing [3], transmission-layer security [3] features and audit-ability [23].

1.1.2 Overview of Diameter Base Protocol

Diameter provides a base protocol that can be extended for various applications to support AAA services. It must follow [12] for the minimum requirements of a AAA protocol. Diameter uses TCP and SCTP to transport messages. In IETF AAA Working Group, they bring

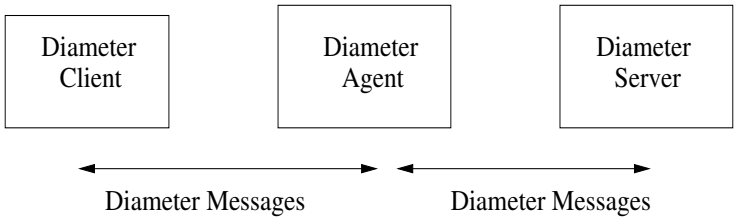

Figure 1.1: Typical Message Flow in Diameter

up some applications those must base on Diameter base protocol, such as EAP and Mobile IP.

Fig. 1.1 illustrates the typical message flow between different Diameter nodes. A Diameter client is a device performing access control at the edge of the network. The client could be a NAS or a Authenticator in EAP. Furthermore any node can initiate a request, Diameter is a peer-to-peer protocol.

In Diameter, the messages exchanged between peers are primarily in the form of Attribute Value Pairs (AVPs) similar to RADIUS. Each Diameter message may contain multiple AVPs required by the application. As mentioned above, We illustrates further detail of above mention in section 2.3 of this thesis. The Diameter base protocol also provides end-to-end security. To secure Diameter messages, Diameter server must support Transport Layer Security (TLS [24]) and IPSec. TLS provides for mutual authentication, integrity-protected ciphersuite negotiation and key exchange between two endpoints. Through the protection of TLS, connections can be established privately and reliably. IPSec provides the security in the IP layer. With IPsec, IP packets transmitted between two entities are protected by the Authentication Header (AH [25]) and ESP. Based on the above techniques, the transport of Diameter protocol must with non-null encryption and authentication algorithms to provide per-packet authentication, integrity and confidentiality. Moreover, for IPSec, the Diameter implementation must support Internet Key Exchange (IKE [26]) for peer authentication, key management and negotiation of security association. For TLS, the certification is required in

order to ensure mutual authentication.

1.1.3 Overview of Diameter EAP Application

The Extensible Authentication Protocol (EAP [27]) provides a standard mechanism for support of various authentication methods. In order to supporting EAP in the Diameter, IETF has started to develop Diameter EAP application which is still a work in progress. The document defines the Command-Codes and AVPs necessary to carry EAP packets within the Diameter protocol between NAS and backend authentication server.

In the Diameter EAP application, two Diameter EAP message types are specified, the Diameter-EAP-Request and Diameter-EAP-Answer. Both of them are set to a value 268 in the Command-Code field and are distinguished by the Diameter flag in Diameter header. The Diameter-EAP-Request message is usually initiated by the client to the server, for requesting the authentication. In contrary, Diameter-EAP-Answer message is initiated by the server for responding correspondent request. All Diameter EAP messages should include an EAP-Payload AVP that contains the EAP information. By encapsulating the EAP within the Diameter messages, EAP authentication can be proceeded between the client and server.

1.2 Overview of IEEE 802.1x

The IEEE 802.1x[10] defines a mechanism for port-based network access control. It base upon the EAP to provide compatible authentication and authorization mechanisms for devices interconnected by IEEE 802 LANs. As depicted in Fig. 1.2, there are three main components in the 802.1x authentication system which is supplicant, authenticator and AAA server respectively. In a WLAN, supplicant is a Mobile Node (MN). AP usually represents an authenticator. AAA server such as RADIUS or Diameter server is an authentication server. The port in 802.11 WLANs represents the association between a MN and an AP.

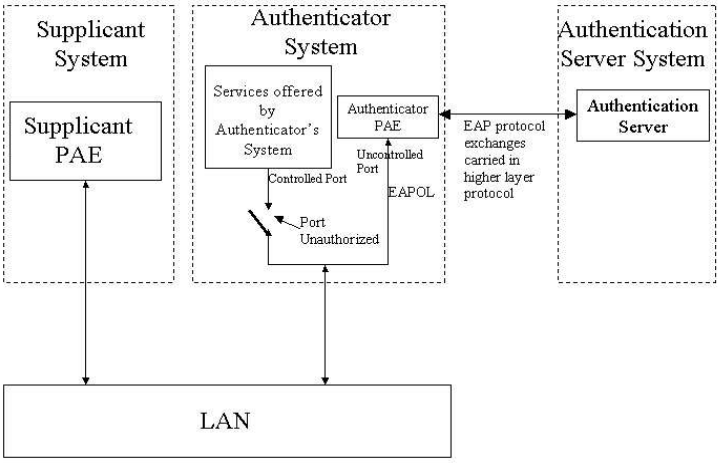

Figure 1.2: Supplicant, Authenticator and AAA Server

Both the supplicant and the authenticator have a Port Access Entity (PAE) that operates the algorithms and protocols associated with the authentication mechanisms. In Fig. 1.2, the authenticator's controlled port is in the unauthorized state, which will block all traffic except 802.1x messages. The authenticator PAE will switch to the authorized state after the MN is authenticated successfully.

It is important to note that 802.1x doesn't provide the actual authentication mechanisms. 802.1x get use of a protocol called EAP both tin wired and wireless LAN and supports multiple authentication methods. Based upon EAP, the 802.1x standard provides a number of authentication mechanisms including Message Digest five (MD5 [28]), TLS, Tunneled TLS Authentication Protocol (TTLS [29]), Protected Extensible Authentication Protocol (PEAP [30]) and others. We will give more discussion on EAP types in next section. 802.1x also defines EAPOL (EAP over LANs) to encapsulate EAP messages between the supplicant and the authenticator over 802 networks. The authenticator PAE relays all EAP messages between supplicant and authentication server.

1.3 Overview of PPP Extensible Authentication Protocol (EAP)

EAP is a extension to PPP. EAP is a general protocol for authentication that also supports multiple authentication methods such as token cards, Kerberos, one-time passwords, certificates, public key authentication and smart cards. It allows negotiation of an Authentication Protocol for authenticating its peer before allowing Network Layer protocols to transmit over the link.

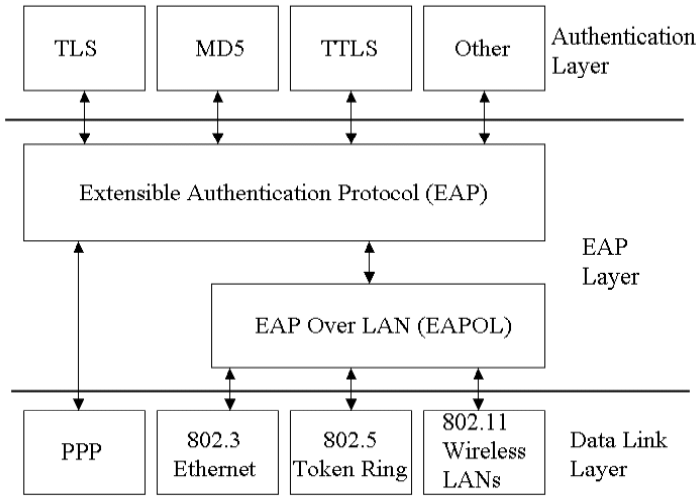

Figure 1.3: Block Diagram of EAP and Associated Layers

WIRE Diameter supports most common authentication methods such as EAP-MD5, EAP-TLS, EAP-TTLS as well as EAP-PEAP. Fig. 1.3 shows that all of these authentication methods are based on EAP. Our EAP state machines also follow [31] to implementation. New authentication mechanism can be added to WIRE Diameter easily. In following context, we will discuss several EAP authentication methods in detail.

1.3.1 EAP Message Digest 5 Authentication Protocol (EAP-MD5)

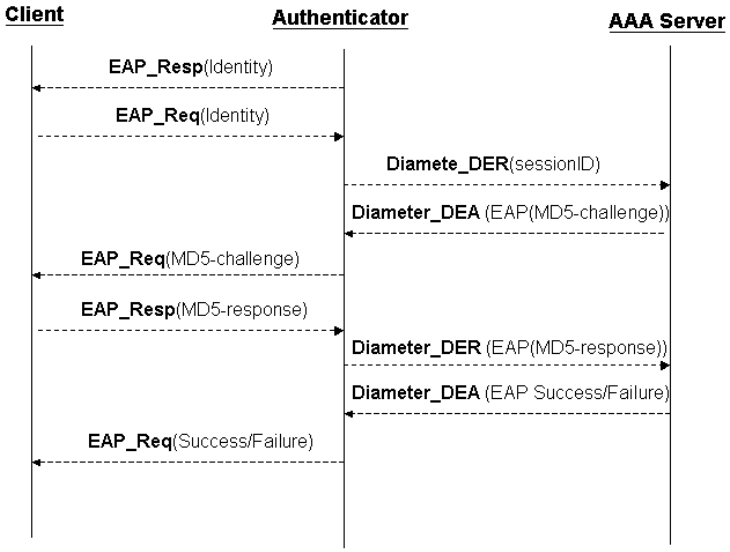

Figure 1.4: EAP-MD5 Authentication Flowchart

EAP-MD5 (EAP-Type = 4), one of most popular EAP types that can be used in the marketplace with convenient. Users simply type in their usernames and passwords to be authenticated. Server authenticates the user through hashing function. This is a simple and reasonable choice for wired LANs where there is low risk of attackers sniffing or active attack. However, EAP-MD5 is not suitable for wireless LANs because attackers can easily sniff station identities and password hashes.

1.3.2 EAP Transport Level Security Authenticated Protocol (EAP-TLS)

EAP-TLS (EAP-Type = 13) provides mutual authentication, integrity-protected ciphersuite negotiation and key exchange between two endpoints. The EAP-TLS conversation typically begin with the authenticator and the peer negotiating EAP. After establishing conversation, peers can securely communicate with in encrypted TLS tunnel.

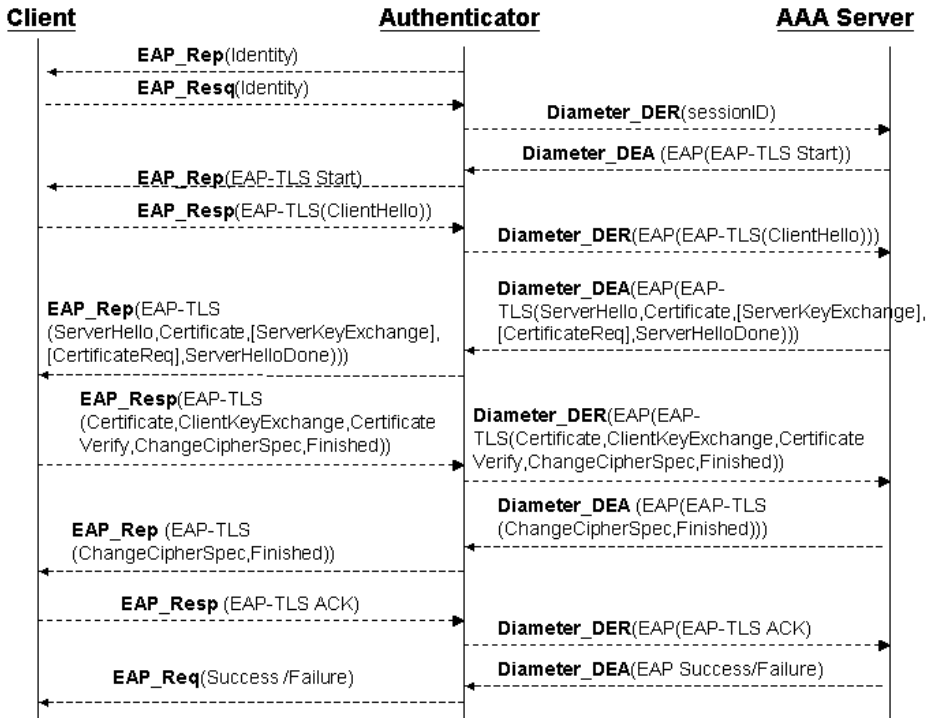

Figure 1.5: EAP-TLS Authentication Flowchart

1.3.3 EAP Tunneled TLS Authentication Protocol (EAP-TTLS)

EAP-TTLS (EAP-Type = 21) extends authentication negotiation by using secure connection which is established by the TLS handshake to exchange additional information between client and server. A EAP-TTLS negotiation comprises two phases: the TLS handshake phase and the TLS tunnel phase. During phase 1, TLS is used to authenticate the TTLS server and the client. In phase 2, EAP-TTLS specifies how user authentication may be performed to encrypting the TLS record layer within the tunnel. The user authentication may be EAP or a legacy protocol such as PPP Authentication Protocols (PAP [32]), PPP Challenge Handshake Authentication Protocol (CHAP [33]), Microsoft PPP CHAP Extensions (MS-CHAP [34]) or Microsoft PPP CHAP Extensions, Version 2 (MS-CHAP-V2 [35]).

1.3.3. EAP Tunneled TLS Authentication Protocol (EAP-TTLS)

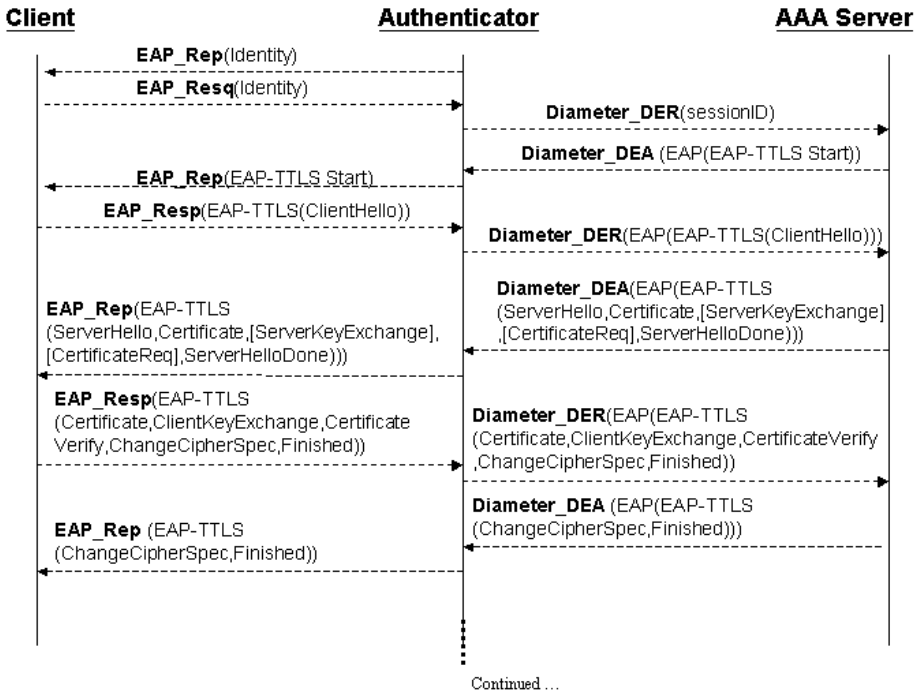

Figure 1.6: The Phase1 of EAP-TTLS Authentication Flowchart

This subsection describes those methods for tunnelling specific authentication protocols within EAP-TTLS. Thus, EAP-TTLS messaging can be described using a layered model, where each layer encapsulates the layer beneath it. The following table clarifies the relationship between protocols.

Carrier Protocol (PPP, EAPOL, RADIUS, Diameter, etc.)
EAP
EAP-TTLS
TLS
User Authentication Protocol (PAP, CHAP, MS-CHAP, etc.)

Table 1.1: EAP-TTLS Protocol Layer Model

- **PAP**

In PAP, the client tunnels USER-NAME(AVP Code = 1) and USER-PASSWORD(AVP

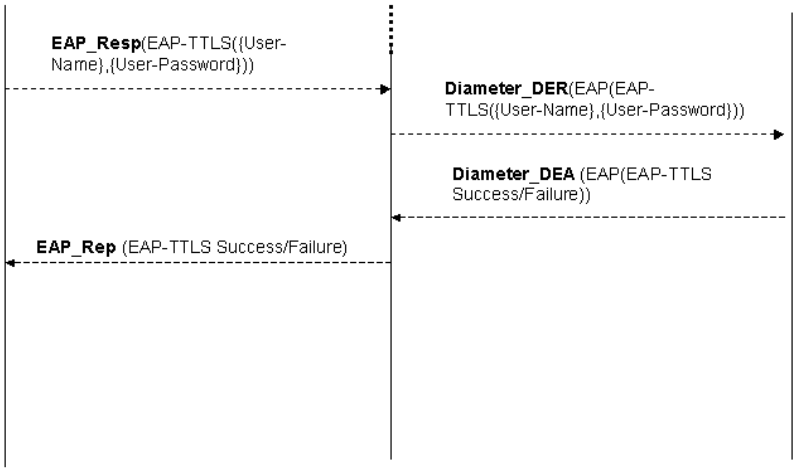

Figure 1.7: The Phase2 of EAP-TLS Authentication Flowchart with PAP

Code = 2) AVPs to the TTLS server. Upon receipt of two AVPs from the client, the TTLS server forwards them to the AAA/H (a AAA server in the user's home domain) in a Access-Request. The AAA/H may immediately respond with an Access-Accept or Access-Reject, the TTLS server then completes the negotiation by sending an EAP-Success or EAP-Failure to the AP using AAA carrier protocol(e.g RADIUS / Diameter protocol).

- **CHAP**

  In CHAP, the client tunnels USER-NAME(AVP Code = 1), CHAP-CHALLENGE(AVP Code = 60) and CHAP-PASSWORD (Assuming AVP Code = 3) AVPs to the TTLS server. Upon receipt of these AVPs from the client, the server must verify whether the value of CHAP-CHALLENGE AVP and CHAP Identifier in the CHAP-PASSWORD AVP are equal to the values generated as challenge material. The AAA/H may im-

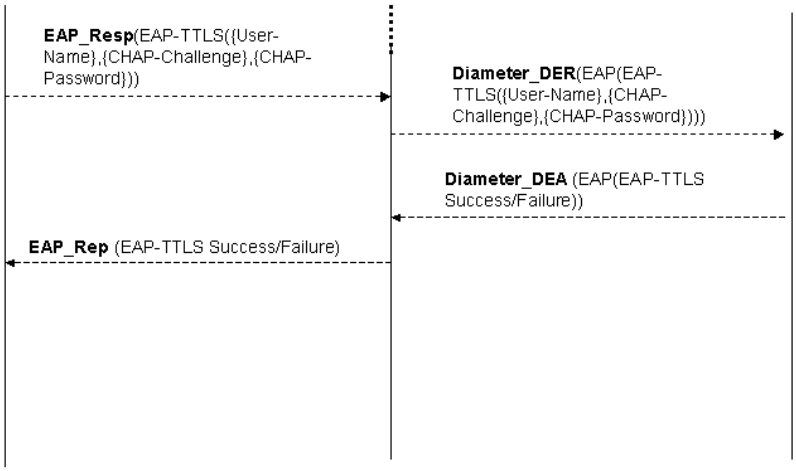

Figure 1.8: The Phase2 of EAP-TLS Authentication Flowchart with CHAP

mediately respond with an Access-Accept or Access-Reject, the TTLS server then completes the negotiation by sending an EAP-Success or EAP-Failure to the AP using AAA carrier protocol.

- **MS-CHAP**

  In MS-CHAP, the client tunnels MS-VENDOR-ATTR(AVP Code = 311), MS-CHAP-CHALLENGE(AVP Code = 11) and MS-CHAP-RESPONSE(AVP Code = 1) AVPs to the TTLS server. Upon receipt of these AVPs from the client, the server must verify that value of the MS-CHAP-CHALLENGE AVP and the value of the Ident in the client's MS-CHAP-RESPONSE AVP are equal to the values generated as challenge material. The AAA/H may immediately respond with an Access-Accept or Access-Reject, the TTLS server then completes the negotiation by sending an EAP-Success or EAP-Failure to the AP using AAA carrier protocol.

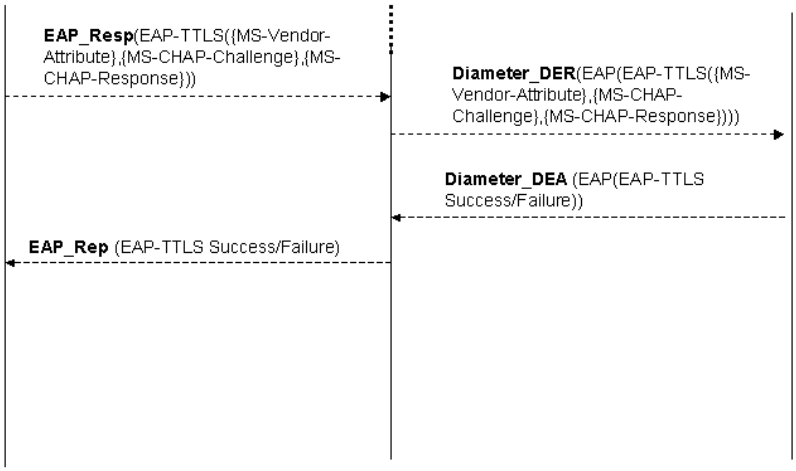

Figure 1.9: The Phase2 of EAP-TLS Authentication Flowchart with MS-CHAP

- **MS-CHAP-V2**

  In MS-CHAP-V2, the client tunnels USER-NAME(AVP Code = 1), MS-CHAP-CHALLENGE(AVP Code = 11) and MS-CHAP2-RESPONSE(AVP Code = 25) AVPs to the TTLS server.Upon receipt of these AVPs from the client, the server must verify that value of the MS-CHAP-CHALLENGE AVP and the value of the Ident in the client's MS-CHAP2-RESPONSE AVP are equal to the values generated as challenge material. If the authentication is successful, the AAA/H will respond with an Access-Accept containing the MS-CHA2-SUCCESS attribute. This attribute contains a 42-octet string that authenticates the AAA/H to the client based on the *Peer-Challenge*. Upon receipt of the MS-CHAP2-SUCCESS AVP, the client is able to authenticate the AAA/H. If the authentication succeeds, the client sends an EAP-TTLS packet to the TTLS server containing no data as a acknowledgement (ACK for short). Upon re-

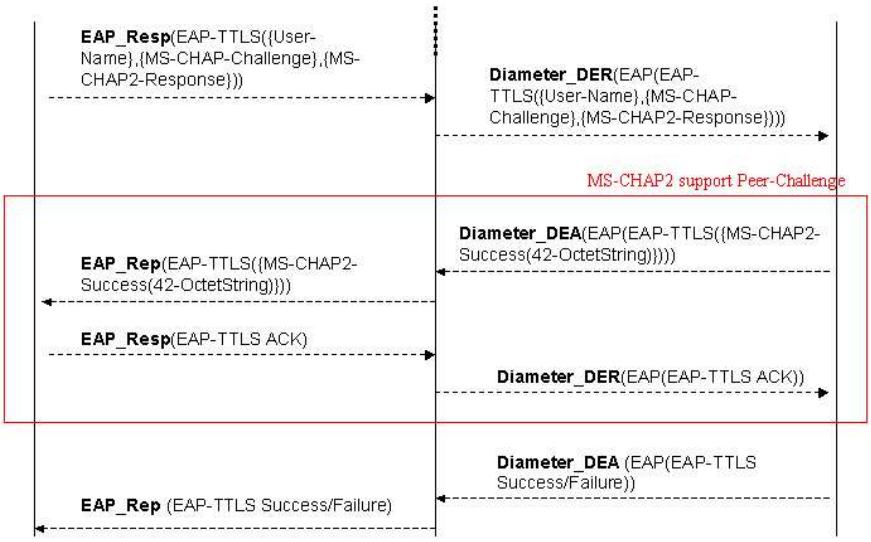

Figure 1.10: The Phase2 of EAP-TLS Authentication Flowchart with MS-CHAP-V2

ceipt of the empty EAP-TTLS packet from the client, the TTLS server now issues an EAP-Success. However, current client and server product in the market do not support Peer-Challenge in MS-CHAP-V2. So If the authentication is successful, the AAA/H will respond with an Access-Accept without the MS-CHA2-SUCCESS attribute. WIRE Diameter also support Peer-Challenge / non-Peer-Challenge in MS-CHAP2-SUCCESS.

1.3.4 Protected EAP Protocol (PEAP)

PEAP (EAP-Type = 25) is proposed by Microsoft, Cisco and RSA Security to securely transport authentication data. PEAP provides an encrypted and authenticated tunnel based on TLS that encapsulates EAP authentication mechanisms. PEAP uses TLS to protect against

rogue authenticators and various attacks on confidentiality and integrity. Generally, both of TTLS and PEAP use TLS to negotiate an end-to-end tunnel in phase 1. Then phase 2, within the TLS session, zero or more EAP methods are carried out.

The evolution of PEAP have EAP-PEAPv0[36], EAP-PEAPv1[30] and EAP-PEAPv2[37]. We use the version of EAP-PEAP to classify PEAP-supported supplicants. WIRE1x and Open1x are adopt EAP-PEAPv0 and EAP-PEAPv1. WIRE Diameter support EAP-PEAPv1, because we did not found any supplicant product follows EAP-PEAPv2. To end of this section, we summarize by the Table 1.2 in presenting comparison of EAP methods.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows.

- Chapter 2 describes the software architecture of WIRE Diameter including programming language, platform and necessary libraries.
- Chapter 3 discuss related state machines in the WIRE Diameter containing authenticator state machine, passthrough state machine, backend adapter state machine and a lot of EAP method state machines.
- Chapter 4 present the testbed of WIRE Diameter.
- Chapter 5 summarize this thesis.

	EAP-MD5 (RFC 2284)	EAP-TLS (RFC 2716)	TTLS (Internet draft-04, April 2004)	PEAP (Dead, Internet draft-07, Ver2, Oct 2003
Authentication methods	Password Hash	Client certificates	Any	Any EAP method
Basic protocol structure	Verify Challenge-Value	Establish TLS session and validate certificates on both client and server	(1) Establish TLS session (2) Exchange AVP of TLS tunnel between client and server	(1) Establish TLS session (2) Authenticate phase2 method through protected EAP
Server Certificate	None	Required	Required	Required
Client Certificate	None	Required	Optional	Optional
Protection of user identity exchange	No	No	Yes; protected by TLS	Yes; protected by TLS
Authentication direction		Mutual: Uses digital certificates both ways	Mutual: Certificate for server authentication, and tunneled method for client	Mutual: Certificate for server authentication, and protected EAP method for client

Table 1.2: Comparison of EAP Methods

Chapter 2

WIRE Diameter Software Architecture

The WIRE Diameter is an open source implementation of Diameter Protocol and IEEE 802.1x authenticator, developed by Wireless Internet Research & Engineering (WIRE) Laboratory, sponsored by ITRI. The architecture of WIRE Diameter implementation use lots of object-oriented design patterns for maintainability and scalability. In this chapter, we will describe relative essentials about this software.

2.1 Overview of WIRE Diameter

This section elucidate some decisions during the design of WIRE Diameter. It is important to note that WIRE Diameter must based upon established rules of software engineering as well as the current trend specified in the [4] and [3]. WIRE Diameter is based on OpenDiameter to extend its authentication methods. Most of this software architecture is inherited from OpenDiameter.

2.1.1 Programming Language

The programming language for WIRE Diameter is C++. It has the advantage of object oriented, widespread familiarity and easy maintenance. In addition, WIRE Diameter utilizes standard C++ libraries to allow for speed of development.

2.1.2 Platform and System Support

WIRE Diameter was development to keep platform independent as possible. All system calls are abstracted by an internal utilities layer that provide a platform independent interface to the core diameter logic. So current WIRE Diameter support Linux, FreeBSD and Windows in OS.

2.2 Open Source Libraries

In this section, we introduce several open libraries that are applied in WIRE Diameter. We also describe their characteristics and applications.

2.2.1 OpenDiameter

The OpenDiameter is a open-sources for the Diameter base protocol. This library provide standard implementation of Diameter base protocol and some Diameter applications. The architecture of the OpenDiameter borrows heavily from the design patterns developed in Adaptive Communication Environment (ACE[38]). In particular, the socket acceptor, connector and thread pooling patterns are employed. In addition to using ACE based patterns, the OS abstraction layer provided by the ACE library is heavily utilized in this implementation.

OpenDiameter Library Modules

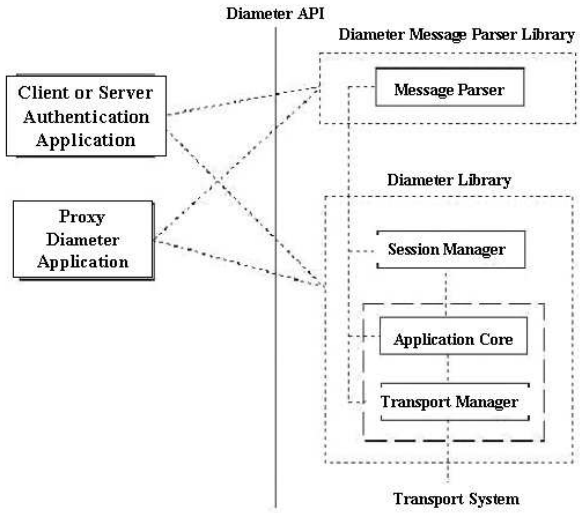

Figure 2.1: OpenDiameter Library Modules

The current architecture of OpenDiameter software is divided into four logical modules, including application core, session manager module, transport manager module and message parser module. As shown in Fig. 2.1, the relationship between logical modules and applications is illustrated. The libdiamparser is implemented as a separate library from the libdiameter. Both of them are common to client or server authentication application.

- **Application Core**

  It is a central storage of all global data and used for initialization and termination services for the entire Diameter library. Included in the global data store of the application core are all the static data contained in the XML configuration files.

- **Transport Manager**

  It is an integral partner of the application core. Its main responsibility is to maintain

connection states with other Diameter peers as well as routing and delivery of Diameter messages. ACE communications acceptor / connector pattern.

- **Session Manager**

  The module is responsible for storing and maintaining Diameter sessions for client / server authentication application using the Diameter library.

- **Message Parser**

  The module is used for parsing Diameter messages including header and payload, where the payload part is consist of Diameter AVPs.

Threaded View of OpenDiameter

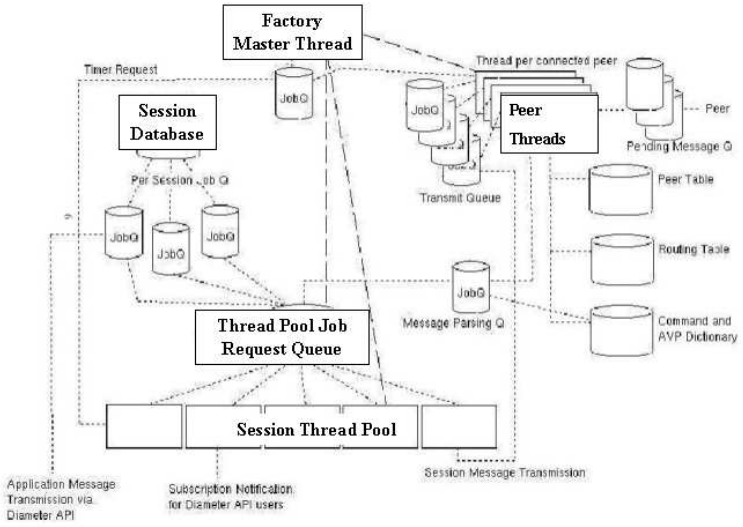

Figure 2.2: Thread View of OpenDiameter

In Fig. 2.2, the execution context for Diameter protocol functions are provided by multiple threads. The thread handling are based on the ACE thread implementations. The issue of contention and serialization between components within the implementation is solved by

using protected queues. As with threading, the queue and locks (which provided protection) are also ACE-based. We list some types of threads that exist in this program.

- **Master Thread**

  It also known as the factory, is responsible for spawning a thread for each active peer connection, initiating and monitoring local asynchronous connection request as well as accepting remote connection request. The thread also maintain the existence of a thread pool.

- **Peer Connection Threads**

  It is responsible for adhering to the peer state machines stated in Diameter, sending and receiving Diameter message. Maintaining the peer table and other functions necessary for establishing v peer connection.

- **Thread Pool**

  The thread pool design pattern exists to facilitate the use of load balancing in session management. The pool is composed of a collection of threads that constantly monitors a job request queue as shown in Fig. 2.2.

2.2.2 Adaptive Communication Environment (ACE)

The ACE is a widely-used, open-source, object-oriented toolkit written in C++ that implements core concurrency and networking patterns for communication software. ACE includes many components that simplify the development of communication software, thereby enhancing flexibility, efficiency, reliability and portability.

ACE provides a rich set of reusable C++ wrapper facades and framework components that perform common communication software tasks across a range of OS platforms. The communication software tasks provided by ACE include event demultiplexing and event handler dispatching, signal handling, service initialization, interprocess communication, shared

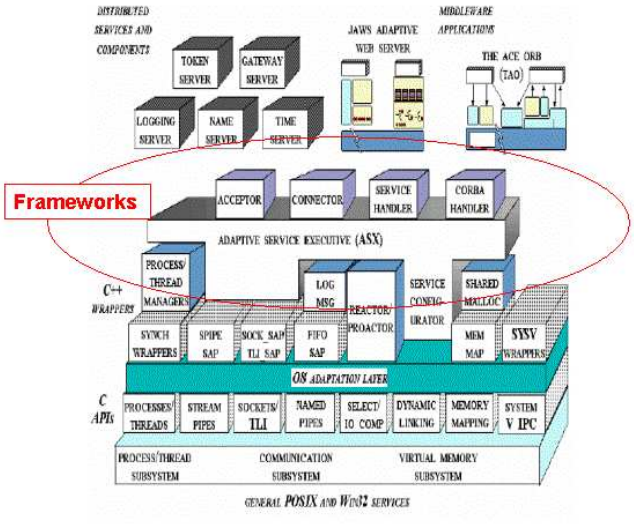

Figure 2.3: The Framework of ACE

memory management, message routing, dynamic (re)configuration of distributed services, concurrent execution and synchronization. In addition, ACE automates system configuration and reconfiguration by dynamically linking services into applications at run-time and executing these services in one or more processes or threads. Therefore, WIRE Diameter use a large number of design patterns developed in ACE through OpenDiameter library.

Fig. 2.3 shows us the architecture of ACE. The ACE Framework Components involve *Event Handing Framework, Connection and Service Initialization Components, Stream Framework and Service Configuration Framework*. In WIRE Diameter, We use *Event Handing Framework* as Reactor, the Reactor provides code for efficient event de-multiplexing and dispatching, which decouples that event de-multiplexing and dispatching, which decouples the event de-multiplexing and dispatch code from the handling code, thereby enhancing reusability and flexibility. Moreover, We also use much concept of *Connection and Service Initialization Components* as Connector and Acceptor components that decouple the initia-

tion of a connection from the service performed by the application after the connection has been established.

2.2.3 Xerces

Xerces-C++ [39] is a validating XML parser written in a portable subset of C++. A shared library is provided for parsing, generating, manipulating, and validating XML documents. The parser provides high performance, modularity, and scalability. Source code, samples and API documentation are provided with the parser.

Xerces has rich generating and validating capabilities. The parser is used for: Building XML-savvy Web servers, Building next generation of vertical applications that use XML as their data format ,On-the-fly validation for creating XML editors, Ensuring the integrity of e-business data expressed in XML ,Building truly internationalized XML applications.

WIRE Diameter employs Xerces to parser it's configuration file. The configuration file can be modify to indicate different parameters within WIRE Diameter.

2.2.4 OpenSSL

The OpenSSL [40] is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson. The OpenSSL toolkit is licensed under an Apache-style licence, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions. This Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and TLS v1 protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

WIRE Diameter utilizes OpenSSL to encrypt and decrypt the TLS record in the authentication procedure. OpenSSL also be used to verify and generate certificate.

2.2.5 Pcap / WinPcap

Pcap [41]is an architecture for packet capture and network analysis for the Linux and FreeBSD platforms. It includes a kernel-level packet filter, a low-level dynamic link library (packet.dll), a high-level and system-independent library (libpcap.dll). The packet filter is a device driver that ability to capture and send raw data from a network card, with the possibility to filter and store in a buffer the captured packets.

WinPcap [42] is porting from pcap library for the Win32 platforms. It must modify it system-independent library (wpcap.dll, based on libpcap version 0.6.2). Let the packet filter of WinPcap adds to Windows 95, 98, ME, NT, 2000, XP and 2003.

WIRE Diameter employs Pcap/WinPcap to transmit layer 2 frames in authenticator.

2.2.6 Libnet

Libnet [43] is a generic networking API that provides access to several protocols. It is not designed as a 'all in one' solution to networking. Many features that are common in some network protocols are not available with Libnet, such as streaming via TCP/IP. We feel that Libnet should not provide specific features that are possible in other protocols. If we restrict Libnet to the minimal needed to communicate (datagram/packets) then this allows it to support more interfaces.

In Authenticator, the WIRE Diameter must prepare layer 2 frames and deliver those one. So we use Libnet to get the MAC(Medium Access Control) address of authenticator interface.

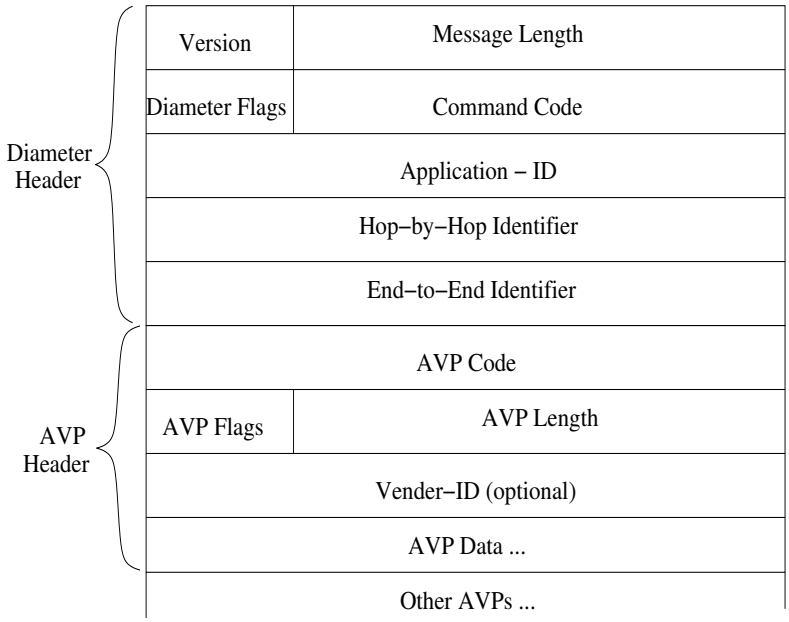

Figure 2.4: The Diameter Message Format

2.3 WIRE Diameter Message Processing

In this section, we introduce the message processing of Diameter base protocol. The [3] provides the delivery of AVPs, capabilities negotiation, error notification, extensibility and basic services necessary for applications.

2.3.1 Diameter Message and Attribute Value Pair (AVP) Format

AVPs are data structures used to carry information. The format of Diameter message is depicted in Fig. 2.4, where AVPs are appended after the Diameter header. Each Diameter message may contain multiple AVPs required by the application. And each AVP carries specific data indicated by the AVP Code filed in AVP header for specific usage. For example, the initial request message and all subsequent messages from a client would include the Session-Id AVP to identify the sessions. The Result-Code AVP is included in the answer

message to indicate success or failure of the request. These AVPs included in Diameter messages are determined by the applications. One can easily extend Diameter service by defining and creating new AVPs or applications. This makes the protocol more extensible to incorporate with different applications.

The [3] also describe a complete Diameter message, include of Diameter header and a lot of Diameter AVPs. Diameter AVPs carry specific authentication, accounting, authorization, routing and security information as well as configuration details for the request and reply.

2.3.2 Standard Diameter Base Protocol Procedure

Every Diameter message must contain a command code in its header's Command-Code field, which is used to determine the action that is to be taken for a particular message. The following Command Codes are defined in [3] and are applied to WIRE Diameter:

- **Capabilities-Exchange-Request/Answer (CER/CEA)**
  CER/CEA (Command Code = 257), Diameter peers establish a transport connection. This message allows the discovery of a peer's identity and its capabilities(protocol version number, supported Diameter applications, security mechanisms, etc.). A receiver of a CER message that does not have any applications in common with the sender MUST return a CEA with the Result-Code AVP set to DiameterNoCommonApplication (AVP Code = 268), and should disconnect the transport layer connection. (Section 5.3.1 of [3])

- **Re-Auth-Request/Answer (RAR/RAA)**
  RAR/RAA (Command Code = 258), a Diameter server may initiate a re-authentication and/or re-authorization service for a particular session by issuing a RAR. An access device that receives a RAR message with Session-Id equal to a currently active session must initiate a re-auth towards the user, if the service supports this particular feature.

Each Diameter application must state whether service-initiated re-auth is supported, since some applications do not allow access devices to prompt the user for re-auth (Section 8.3.1 of [3]). A RAR and RAA message must be followed by an application-specific authentication and/or authorization message, as Fig. 2.5 and Fig. 2.6.

- **Session-Termination-Request/Anwer (STR/STA)**
  STR/STA (Command Code = 270), When a user terminate this session(e.g send EAPOL-Logoff) , shutdown the access device or receives arrival ASR, the access device must handle this situation. When Diameter authorization terminates, the access device that provided the service must issue a STR message to the Diameter server that authorized the service, to notify it that the session is no longer active. (Section 8.4.1 of [3])

- **Abort-Session-Request/Answer (ASR/ASA)**
  ASR/ASA (Command Code = 274), a Diameter server may request that the access device stop providing service for a particular session. An access device that receives a RAR message with Session-Id equal to a currently active session must initiate a re-auth towards the user, if the service supports this particular feature. (Section 8.3.1 of [3])

- **Device-Watchdog-Request/Answer (DWR/DWA)**
  DWR/DWA (Command Code = 280), it is used to detect those transport failures as soon as possible. Detecting such failures will minimize the occurrence of messages sent to unavailable agents, resulting in unnecessary delays, and will provide better Failover performance. (Section 5.5.1 of [3])

2.3.3 Standard Diameter EAP Procedure

Peers will communicate with DER/DEA and RAR/RAA for EAP application. We present the Diameter EAP message and several Diameter AVPs for this application in the following.

```
<Re-Auth-Request> ::=
                    < Diameter Header: 258, REQ, PXY >
                    < Session-Id >
                    { Auth-Application-Id }
                    { Origin-Host }
                    { Origin-Realm }
                    [ Destination-Host ]
                    { Destination-Realm }
                    { Auth-Request-Type }
```

Figure 2.5: Re-Auth-Request (RAR) Message Format

Diameter-EAP-Request/Answer (DER/DEA)

DER/DEA (Command Code = 268), Diameter peers convey an EAP Request / Response with DER/DEA. The DER must contain one EAP-Payload AVP, which contains the actual EAP payload. An EAP-Payload AVP with no data may be sent to the Diameter server to initiate an EAP authentication session. The DER message MAY be the result of a multi-round authentication exchange, which occurs when the DEA is received with the Result-Code AVP set to DiameterMultiRoundAuth [3] (Section 3 of [4]).

In WIRE Diameter, the DER/DEA message contains several AVPs, include of Session-Id, Auth-Application-Id, Auth-Request-Type, Result-Code, Origin-Host, Origin-Realm, Destination-Host, and Destination-Realm AVPs (these are defined in [3]). Furthermore, [4] also define new AVPs for Diameter EAP application, as *EAP-Payload AVP*. Those are defined in Fig. 2.7 and Fig. 2.8.

- **EAP-Payload**

```
<Re-Auth-Answer> ::=
                    < Diameter Header: 258, REQ, PXY >
                     < Session-Id >
                     { Auth-Application-Id }
                     { Result-Code }
                     { Origin-Host }
                     { Origin-Realm }
```

Figure 2.6: Re-Auth-Answer (RAA) Message Format

AVP Code: 2000 (Assumption)

AVP Data Format: OctetString

Description: The AVP is used to encapsulate the actual EAP packet that is being exchanged between the EAP client and the home Diameter server.

2.3.4 WIRE Diameter Authentication Procedure

First, WIRE Diameter must communicate between Diameter server and Diameter NAS (also names as Authenticator) to create a Diameter session channel by Diameter message, such as CER/CEA, WDR/WDA, STR/STA, ARR/ARA. The peers will communicate with DER/DEA and RAR/RAA for EAP application. Those Procedures are depicted in Fig. 2.9, which authentication method is MD5 in this example. The Diameter NAS will be responsible for encapsulating EAP packet into DER and forwarding message between user and server.

```
<Diameter-EAP-Request> ::=
        < Diameter Header: 268, REQ, PXY >
        < Session-Id >
        { Auth-Application-Id }
        { Origin-Host }
        { Origin-Realm }
        [ Destination-Host ]
        { Destination-Realm }
        { Auth-Request-Type }
        { EAP-Payload }
```

Figure 2.7: Diameter-EAP-Request (DER) Message Format

```
<Diameter-EAP-Answer> ::=
            < Diameter Header: 268, PXY >
            < Session-Id >
            { Auth-Application-Id }
            { Result-Code }
            { Origin-Host }
            { Origin-Realm }
            [ Destination-Host ]
            { Destination-Realm }
            { Auth-Request-Type }
            { EAP-Payload }
```

Figure 2.8: Diameter-EAP-Answer (DEA) Message Format

2.3.4. WIRE Diameter Authentication Procedure

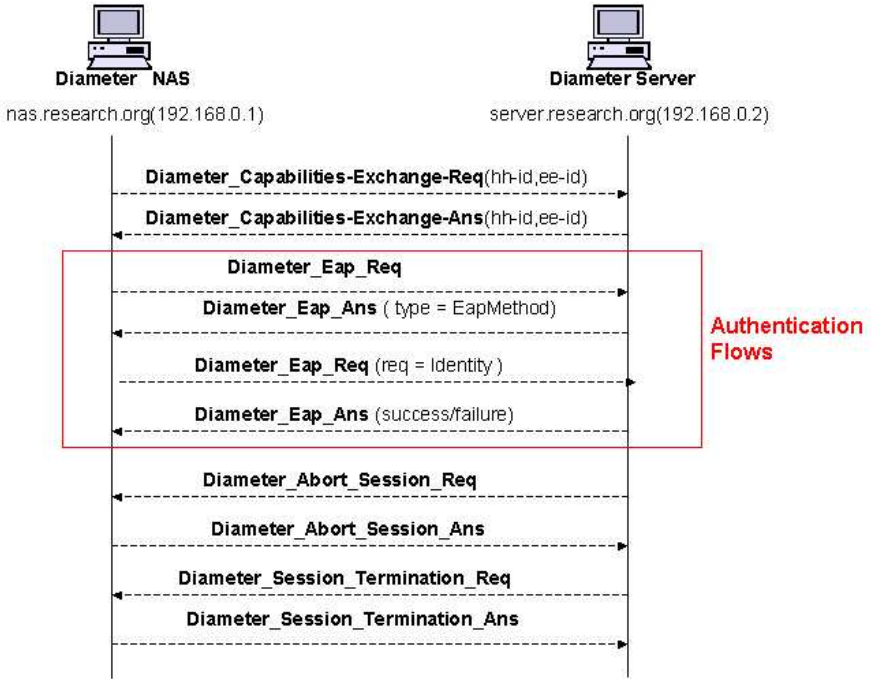

Figure 2.9: Authentication Message Flow in Diameter EAP

Chapter 3

WIRE Diameter EAP State Machines

[31] describes a set of state machines for EAP peer, EAP authenticator and EAP backend authenticator (for use on AAA servers). These state machines shows how EAP can be implemented to support deployment in AAA architecture. The peer state machine is a illustration of how the EAP protocol defined in [27]-bis may be implemented. However, WIRE Diameter doesn't need peer state machine. The backend and pass-through authenticators illustrate how EAP protocol support defined in [16] may be implemented.

This [31] also describes a state machine based on an EAP Switch model. This model includes interaction of events and actions between the EAP Switch and EAP methods. Implementations may achieve the same results using different techniques.

3.1 Notational Conventions used in State Diagrams

The following state diagrams have been completed based on the conventions specified in [44], section 8.2.1. The complete statements is cited here:

State diagrams are used to represent the operation of the protocol by a number of cooperating state machines each comprising a group of connected, mutually exclusive states.

Only one state of each machine can be active at any given time.

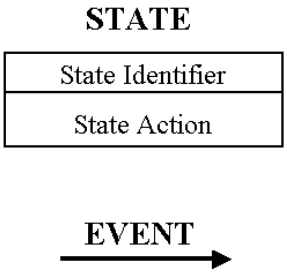

Figure 3.1: Notations in State Diagrams

Each state is represented in the state diagram as a rectangular box, divided into two parts by a horizontal line. The upper part contains the state identifier, written in upper case letters. The lower part contains any procedures that are executed on entry to the state,Such as Fig. 3.1 - (a).

All permissible transitions between states are represented by arrows, the arrowhead denoting the direction of the possible transition. Labels attached to arrows denote the condition(s) that must be met in order for the transition to take place. All conditions are expressions that evaluate to TRUE or FALSE; if a condition evaluates to TRUE, then the condition is met. The label UCT denotes an unconditional transition (i.e., UCT always evaluates to TRUE). A transition that is global in nature (i.e., a transition that occurs from any of the possible states if the condition attached to the arrow is met) is denoted by an open arrow. On entry to a state, the procedures defined for the state (if any) are executed exactly once, as Fig. 3.1 - (b).

On completion of all of the procedures within a state, all exit conditions for the state (including all conditions associated with global transitions) are evaluated continuously until one of the conditions is met. The label ELSE denotes a transition that occurs if none of the other conditions for transitions from the state are met.

Where it is necessary to split a state machine description across more than one diagram, a transition between two states that appear on different diagrams is represented by an exit arrow drawn with dashed lines, plus a reference to the diagram that contains the destination state.

Should a conflict exist between the interpretation of a state diagram and either the corresponding global transition tables or the textual description associated with the state machine, the state diagram takes precedence.

3.2 EAP Switch State Machine

An EAP authentication consists of one or more EAP methods in sequence followed by an EAP Success or EAP Failure sent from the authenticator to the peer. One EAP method must be success to transiting to next authentication method. The EAP Switches state machine takes control of authentication method transition by a sequences of EAP methods. It will swap method after previous one is successfully authenticated. The client was recognized authenticated after it pass a sequence of authentication method.

The use of a particular Passthrough method is a trick to implementing method transition in EAP authenticator. The Passthrough Method appears to Authenticator Switch as a authentication method, but its function is only to Pass EAP messages to a Backend Server where the real Authentication Method resides. [31] includes a state machine for a Passthrough method and this diagram depicts flows between an Authenticator with a Passthrough Method and the Backend accompanied with its Method.

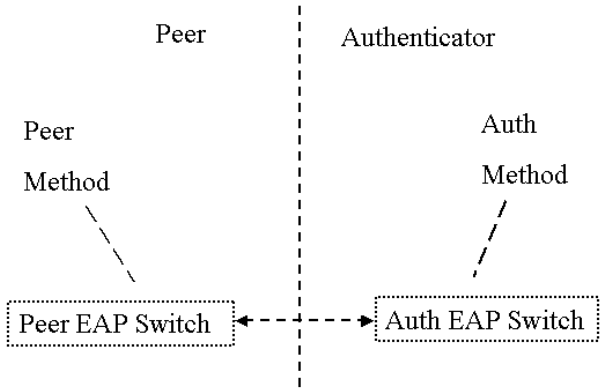

Figure 3.2: Simple EAP Switch Model

WIRE Diameter must implement four components of state machine including Authenticator, Passthrough, Backend-Adapted and some specific method. In Fig. 3.4, the lower layer is taking responsible for link-layer frame transmission. After NAS sending first EAP request and fulfilled its initial authentication methods such like EAP Identify method, the NAS change to passthrough mode which is running with Passthrough state machine. In Passthrough mode, NAS will simply act like a translator which encapsulates client's request in Diameter message and forward to backend server and decapsulated Diameter message from backend server into EAPOL frame to client. This process will continue until backend server signals NAS this client had been admitted to access privilege resource.

The Backend adapter state machine receives EAP responses from NAS inside Diameter EAP messages and passes them to the authenticator state machine, see Fig. 3.5. It also relays the signals returned by the authenticator state machine (e.g eapReq, eapSuccess) to NAS by Diameter EAP messages.

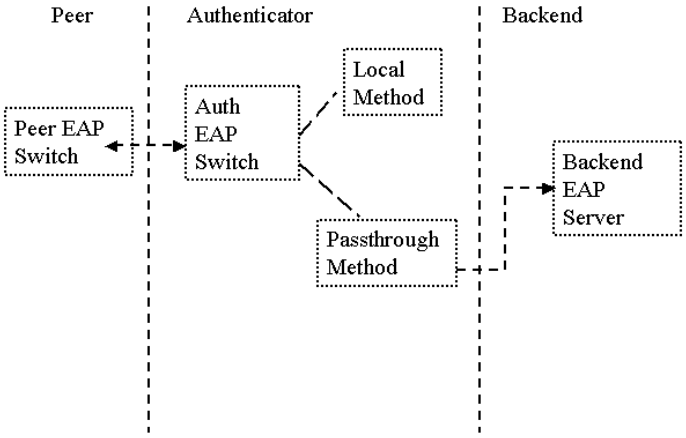

Figure 3.3: Completed EAP Switch Model

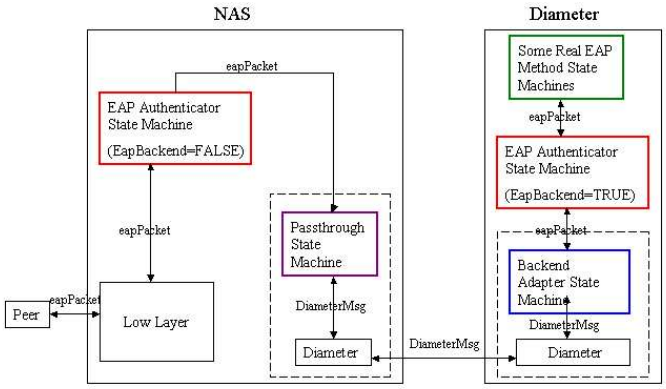

Figure 3.4: State Machine View of WIRE Diameter

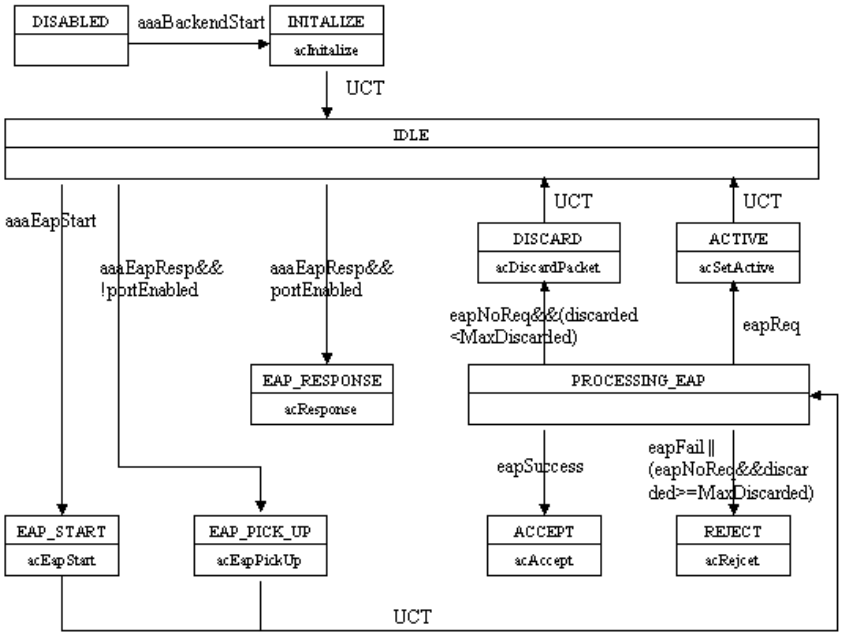

Figure 3.5: Backend Adapter State Machine

3.2.1 Authenticator State Machine

Fig. 3.6 is a diagram of the Authenticator State Machine in [31]. We will declare related variables, functions and states of this state machine, as follow.

- **Local variables**

  Long-term (maintained between packets)

  currentMethod(EAP Type) EAP type, PASSTHROUGH, or NONE.

  currentId(integer) 0-255 or NONE. Usually updated in METHOD state. Indicates the identifier value of the currently outstanding EAP request.

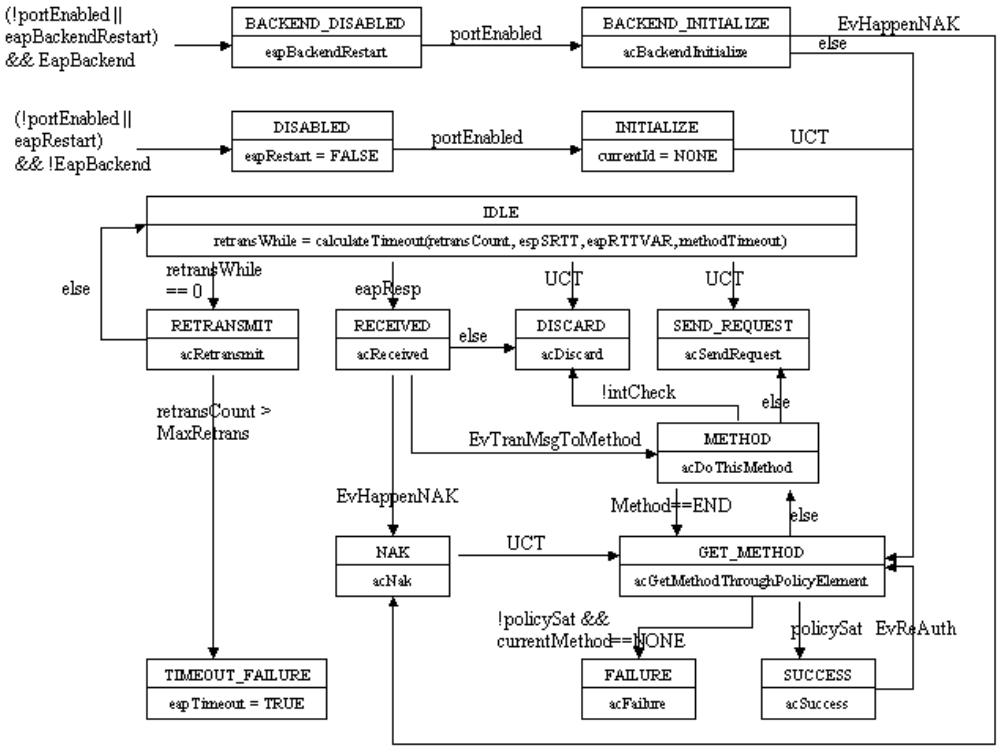

Figure 3.6: Authenticator State Machine

methodState(enumeration) As described above.

retransCount(integer) Set in SEND_REQUEST state. Current number of retransmissions.

lastReqCount(EAP packet) Set in SEND_REQUEST state. EAP packet containing the last sent request.

methodTimeout(integer) Method-provided hint for suitable retransmission timeout, or NONE.

Short-term (not maintained between packets)

rxResp(boolean) Set in RECEIVED state. Indicates the current received packet is an

EAP response.

respId(integer) Set in RECEIVED state. The identifier from the current EAP response.

respMethod(EAP Type) Set in RECEIVED state. The method type of the current EAP response.

succFailDate(EAP Type) Set in METHOD state. Usage described above.

intCheck(boolean) Set in METHOD state. Indicates whether the method has decided to accept the current packet.

policySat(boolean) Set in GET_METHOD state. Stored value of last call to Policy.isSatisfied().

- **Authenticator Procedures - Member Functions**

  parseEapResp() Determine the code, identifier value, and type of the current response. Also checks that the length field is not longer than the Received EAP packet.

  buildSuccess() Create an EAP Success Packet.

  buildFailure() Create an EAP Failure Packet.

  nextId() Determine the next identifier value to use, based on the previous one.

  resetCurrentMethod() Alert the current method that it has been aborted.

  Policy.update() Update all variables related to internal policy state.

  Policy.getNextMethod() Determine the method that should be used at this point in the conversation based on pre-defined policy.

  Policy.isSatisfied() Determine if the policy will allow SUCCESS or not.

  m.integrityCheck() Method-specific procedure to test for the validity of a message.

  m.process() Method procedure to parse and process a response for that method.

  m.buildSuccFail() Only used for the special passthrough method. Constructs a Success or Failure packet based on the EAP payload contained in the AAA accept or reject message.

  m.buildReq() Method procedure to produce the next request.

  m.getNextId() Method procedure that is the parallel of the switch level nextId(). Of-

ten it is up to the method to decide the next ID (particularly in backend authenticators).

m.getKey() Method procedure to obtain key material for use by EAP or lower layers. Also checks that the length field is not longer than the received EAP packet.

- **Authenticator States**

  DISABLED The authenticator is disabled until the port is enabled by the lower layer.

  BACKEND_DISABLED Same for backend server.

  INITIALIZE Initializes variables when the state machine is activated.

  BACKEND INITIALIZE Same for backend server. Also parses the headers of initial response packet (a response to a request sent by the NAS), if any.

  GET_METHOD This state chooses what should happen next: either a method is started, or the conversation is ended.

  IDLE The state machine spends most of its time here, waiting for something to happen.

  RECEIVED This state is entered when an EAP packet is received: the packet header is parsed here.

  METHOD This state builds request packets and processes responses received from the peer. For passthrough mode, see Section 3.2.2.

  SEND REQUEST This state signals the lower layer that a request packet is ready to be sent.

  DISCARD This state signals the lower layer that the response was discarded, and no new request packet will be sent at this time.

  NAK This state processes Nak responses from the peer.

  RETRANSMIT Retransmits the previous request packet.

  SUCCESS A final state indicating success.

  FAILURE A final state indicating failure.

  TIMEOUT FAILURE A final state indicating failure with no EAP Failure packet sent.

3.2.2 Passthrough State Machine

The following is a diagram of the Passthrough State Machine, as Fig. 3.7.Policy starts the (local) Identity method which sends a request packet. When the response comes back, the Identity method processes it and sets methodState=END. Policy then starts the PASSTHROUGH method with methodState PICK-UP-INIT. The PASSTHROUGH method sends the Identity response to the backend server (inside a Diameter Access-Request packet). The backend server responds with some other request (passed PASSTHROUGH method inside a Diameter Access-Challenge), and PASSTHROUGH method relays this request to the peer. This continues until the backend server responds with Access-Accept or Accept-Reject.

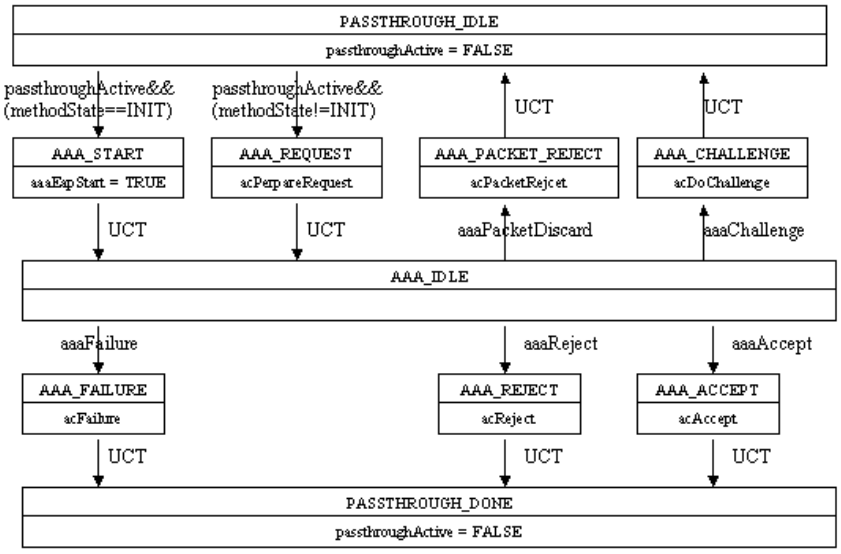

Figure 3.7: Passthrough State Machine

3.3 EAP Method State Machines

Method State Machines are implemented in eapMethodStateMachine from which a distinct class is derived for handling each EAP method. The following methods are defined in the library by WIRE Diameter.

- **Identity**
  Implemented in EapAuthIdentityStateMachine and EapAuthIdentityStateTable_S class by default OpenDiameter.
- **Notification**
  Implemented in EapAuthNotificationStateMachine and EapAuthNotificationStateTable_S class by default OpenDiameter.
- **MD5-Challenge**
  Implemented in EapAuthMD5ChallengeStateMachine and EapAuthMD5ChallengeStateTable_S class by WIRE Diameter, see Section 3.3.1.
- **EAP-TLS**
  Implemented in EapAuthTLSStateMachine and EapAuthTLSStateTable_S class by WIRE Diameter, see Section 3.3.2.
- **EAP-TTLS**
  Implemented in EapAuthTTLSStateMachine and EapAuthTTLSStateTable_ class by WIRE Diameter, see Section 3.3.3.
- **EAP-PEAP**
  Implemented in EapAuthPEAPStateMachine and EapAuthPEAPStateTable_S class by WIRE Diameter, see Section 3.3.4.
- **EAP-PAP**
  Implemented in TTLSPapHandler class by WIRE Diameter, see Section 3.3.3.

- **EAP-CHAP**

  Implemented in TTLSCHAPHandler class by WIRE Diameter, see Section 3.3.3.

- **EAP-MS-CHAP or EAP-MS-CHAP-V2**

  Implemented in TTLSMSChapHandler class by WIRE Diameter, see Section 3.3.3.

  Implemented in PEAPAppAuthHandler class by WIRE Diameter, see Section 3.3.4.

3.3.1 EAP-MD5 Method State Machine

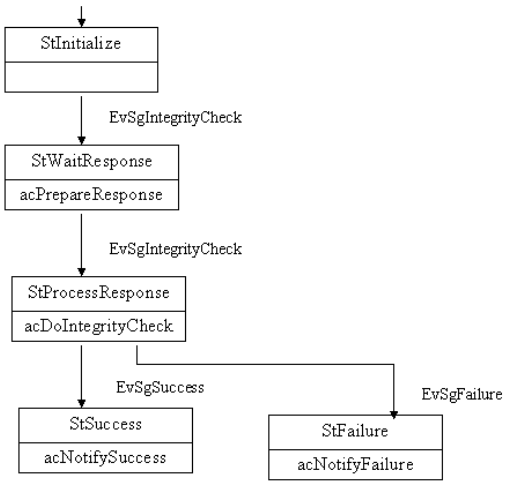

Figure 3.8: EAP-MD5 Method State Machine

Fig. 3.8 is a diagram of the EAP-MD5 Method State Machine. We declare related States, Actions and Events of this state machine, as follow.

- **States of EAP-MD5 Method State Machine**

  StInitialize Initializes variables when the state machine is activated.

StWaitResponse Waiting for EapMethodStateMachine::EvSgIntegrityCheck to happen.

StProcessResponse Handles requests for MD5Challenge method, and builds a response.

StSuccess A final state indicating this method success.

StFailure A final state indicating this method failure.

- **Actions of EAP-MD5 Method State Machine**

  acPrepareRequest Calculate the MS5-Challenge value, Compose this packet and hand over to sending.

  acDoIntegrityCheck Check the MD5-Response value, and Set EvSgSuccess or EvSgFailure for next state.

  acNotifySuccess Update the policy element into PolicyOnSuccess, and generate EvSgEndMethod to notify upper layer.

  acNotifyFailure Update the policy element into PolicyOnFailure, and generate EvSgEndMethod to notify upper layer.

- **Events of EAP-MD5 Method State Machine**

  EvSgSuccess Set in StProcessResponse state.

  EvSgFailure Set in StProcessResponse state.

3.3.2 EAP-TLS Method State Machine

- **States of EAP-TLS Method State Machine**

  StInitialize Initializes variables when the state machine is activated.

  StTlsStart Send TLS-START.

3.3.2. EAP-TLS Method State Machine

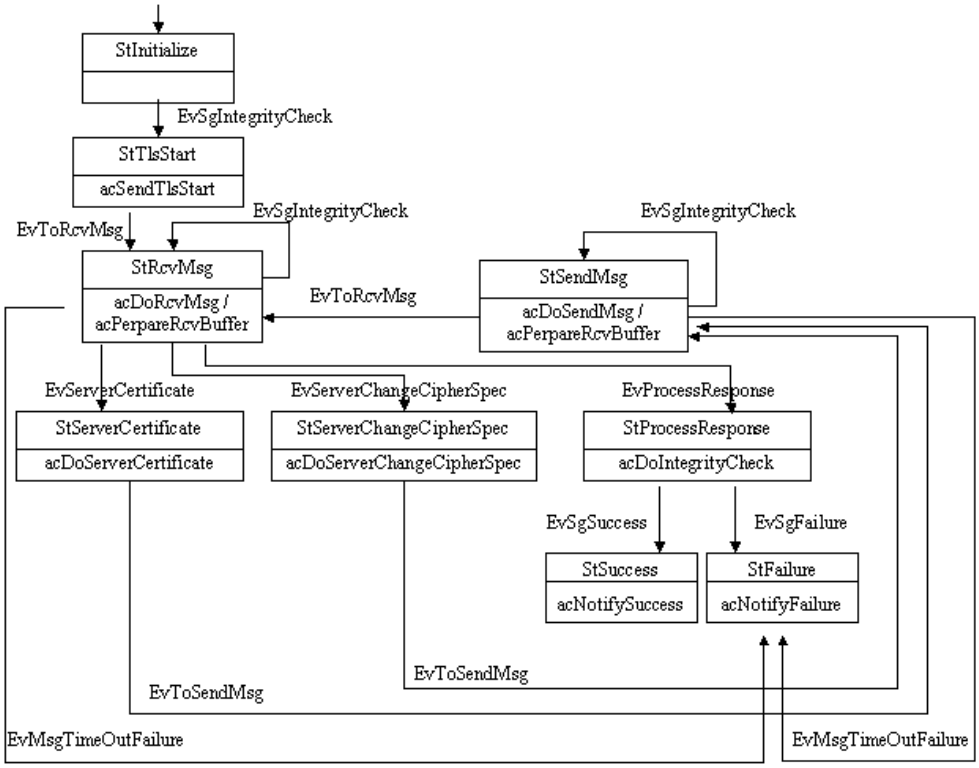

Figure 3.9: EAP-TLS Method State Machine

StRcvMsg This state is entered when an EAP packet is received; the TLS record is parsed here.

StSendMsg This state signals authenticator state machine that a response packet is ready to be sent.

StServerCertificate Do server certificate in the way of EAP-TLS.

StServerChangeCipherSpec Do server change ciper spec in the way of EAP-TLS.

StProcessResponse Handles requests for EAP-TLS method, and builds a response.

StSuccess A final state indicating this method success.

StFailure A final state indicating this method failure.

- **Actions of EAP-TLS Method State Machine**

  acSendTlsStart Send EAP-TLS(Start) to NAS and record previousState.

  acPrepareRcvBuffer Prepare the receiving buffer, When RECEIVED/SEND client messages first time.

  acDoRcvMsg Through GetRxMessage to get message, and record previousState.

  acDoSendMsg Through SetTxMessage to set sent message.

  acDoServerCertificate This action must met succeeded processes. Step1).Get Client Hello from rxTlsRecord, Step2). Generate Server Hello, TLS certificate, [TLS Server_Key _Exchange], [Certificate Request], Server Hello Done, Step3). Write rxTlsRecord to pEapTlsSessionData, Step4).Encrypt EapTlsSessionData with SSL_Read, Step5).Read data from pEapTlsSessionData->pBioFromSSL to txSSLRecord.

  acDoServerChangeCipherSpec Record EAP-TLS(ClientCertificate,...) and Send Change-Cipher-Spec of server.

  acDoIntegrityCheck Deterimate Success or Failure, and Set EvSgSuccess or EvSg-Failure for next state.

  acNotifySuccess Update the policy element into PolicyOnSuccess, and generate EvS-gEndMethod to notify upper layer.

  acNotifyFailure Update the policy element into PolicyOnFailure, and generate EvS-gEndMethod to notify upper layer.

- **Events of EAP-TLS Method State Machine**

  EvSgSuccess Set in StProcessResponse state.

  EvSgFailure Set in StProcessResponse state.

  EvToRcvMsg Set in StTlsStart or StSendMsg state. Indicates the received packet is happen.

  EvToSendMsg Set in StSendMsg/StServerCertificate/StServerChangeCipherSpec state.

Indicates the sent packet is happen.

EvServerCertificate Set in StRcvMsg state. Let process into StServerCertificate.

EvServerChangeCipherSpec Set in StRcvMsg state. Let process into StServerChange-CipherSpec.

EvProcessResponse Set in StRcvMsg state. Let process into StProcessResponse.

EvMsgTimeoutToFailure Set in StRcvMsg or StSendMsg state, when this action is timeout.

3.3.3 EAP-TTLS Method State Machine

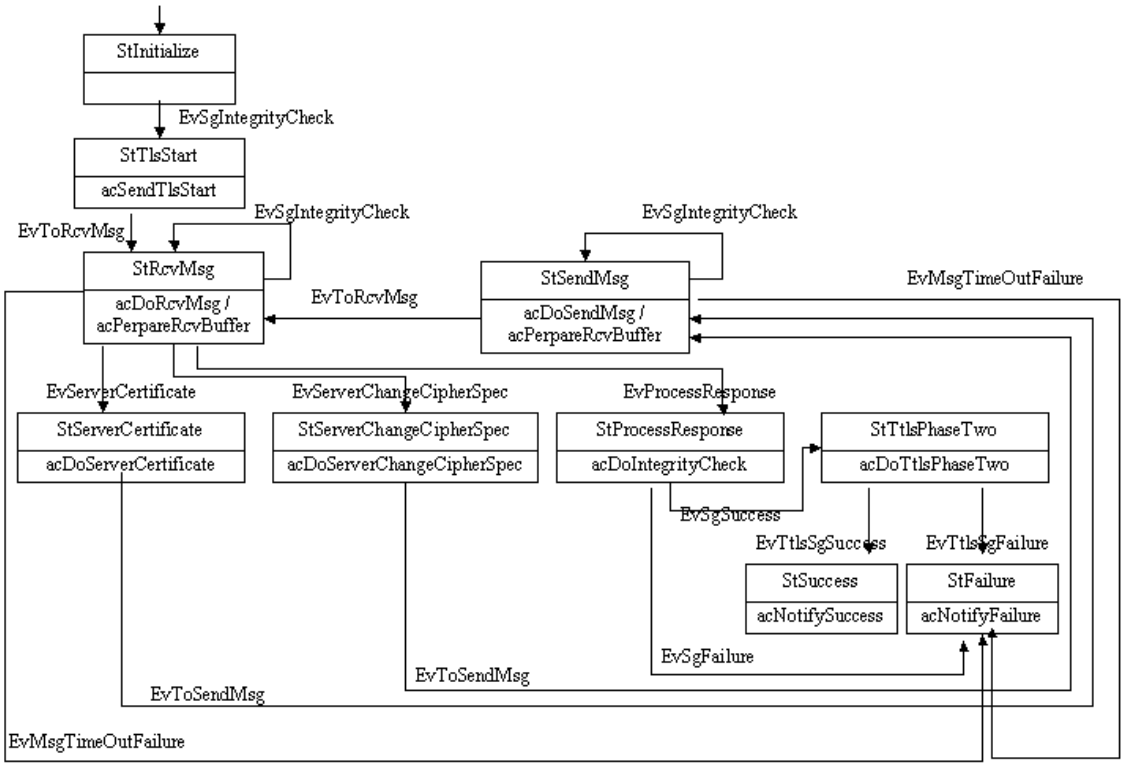

Figure 3.10: Typical EAP-TTLS Method State Machine

3.3.3. EAP-TTLS Method State Machine

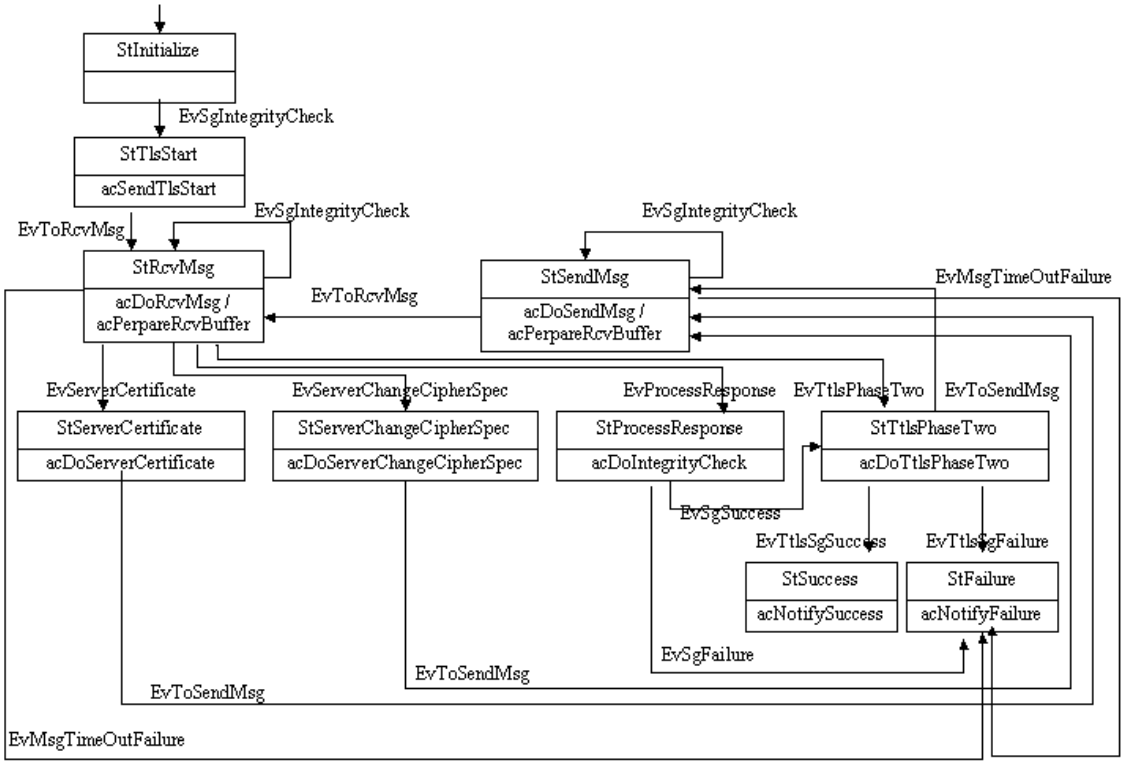

Figure 3.11: EAP-TTLS State Machine, when MS-CHAP-V2 support Peer-Challenge

EAP-TTLS have two phase of authentication, the phase1 is TLS handshake procedure to establish a tunnel, phase2 specifies how user authentication can be performed within established TLS tunnel. In this subsection, we only describe related States, Actions and Events in the phas2 of EAP-TTLS. The WIRE Diameter support Peer-Challenge/non-Peer-Challenge in MS-CHAP-V2. It is more extensible than FreeRadius because FreeRadius not support peer-challenge in MS-Chap-v2. The Peer-Challenge is used to authenticate the opposite side. When the authentication is successful, the AAA/H (AAA Home Server) will respond with Access-Accept containing MS-CHAP2-SUCCESS attribute which contained a 42-octet authentication string to accomplish mutual authentication. This authentication

string is a function output of Peer-Challenge. Client verify the authentication string in MS-CHAP2-SUCCESS attribute to validate server. If this validation is fail, client MUST end this session , otherwise , client must respond with a ACK. Consequently, the MS-CHAP-V2 with Peer-Challenge of EAP-TTLS method state machine is Fig. 3.11, Otherwise is Fig. 3.10.

- **States of EAP-TTLS Method State Machine**

  In phase 1 - TLS Handshake Protocol

  These also defined in Section 3.3.2. StInitialize, StTlsStart, StRcvMsg, StSendMsg, StServerCertificate, StServerChangeCipherSpec, StProcessResponse.

  In phase 2 - TTLS Tunnel Authentication StTtlsPhaseTwo Do tunneled authentication, include of Chap, MS-Chap, MS-Chap-V2, PAP.

  StTtlsSuccess A final state indicating this entire TTLS Authentication is success.

  StTtlsFailure A final state indicating this entire TTLS Authentication is failure.

- **Actions of EAP-TTLS Method State Machine**

  In phase 1 - TLS Handshake Protocol

  These also defined in Section 3.3.2. acSendTlsStart, acPrepareRcvBuffer, acDoRcvMsg, acDoSendMsg, acDoServerCertificate, acDoServerChangeCipherSpec, acDoIntegrityCheck.

  In phase 2 - TTLS Tunnel Authentication

  acDoTtlsPhaseTwo This action have two steps, Step1).Encrypt the TLS SSL Record, those will encapsulate of AVPs within the TLS Record Layer. Step2).Determine this authenticated method in the phase2 with receive AVP type.

  acNotifyTtlsSuccess Update the policy element into PolicyOnSuccess, and generate EvSgEndMethod to notify upper layer.

  acNotifyTtlsFailure Update the policy element into PolicyOnFailure, and generate EvSgEndMethod to notify upper layer.

- **Events of EAP-TTLS Method State Machine**

  In phase 1 - TLS Handshake Protocol

  These also defined in Section 3.3.2. EvSgSuccess, EvSgFailure, EvToRcvMsg, EvToSendMsg, EvServerCertificate, EvServerChangeCipherSpec, EvProcessResponse, EvMsgTimeoutToFailure.

  In phase 2 - TTLS Tunnel Authentication

  EvTtlsSgSuccess Set in StTtlsPhaseTwo state. Let process into StTtlsSuccess.

  EvTtlsSgFailure Set in StTtlsPhaseTwo state. Let process into StTtlsFailure.

  EvTtlsPhaseTwo Set in StProcessResponse state. Let process into StTtlsPhaseTwo.

  If MS_CHAPV2_SUPPORT_PeerChallenge is TRUE, then ...

  EvTtlsPhaseTwo Set in StProcessResponse or StRcvMsg state. Let process into StTtlsPhaseTwo or StSendMsg.

3.3.4 EAP-PEAP Method State Machine

EAP-PEAP have two phase of authentication, the phase1 is typical TLS authentication handshake, the phase2 specifies how user authentication may be performed encrypting the PEAP message within the tunnel. In this subsection, we only describe related States, Actions and Events in the phas2 of EAP-PEAP. The WIRE Diameter support peer-challenge/non-peer-challenge in MS-Chap-v2, more extensible than FreeRadius, because the FreeRadius only support non-peer-challenge in MS-Chap-v2.

- **States of EAP-PEAP Method State Machine**

  In phase 1 - TLS Handshake Protocol

  These also defined in Section 3.3.2. StInitialize, StTlsStart, StRcvMsg, StSendMsg, StServerCertificate, StServerChangeCipherSpec, StProcessResponse.

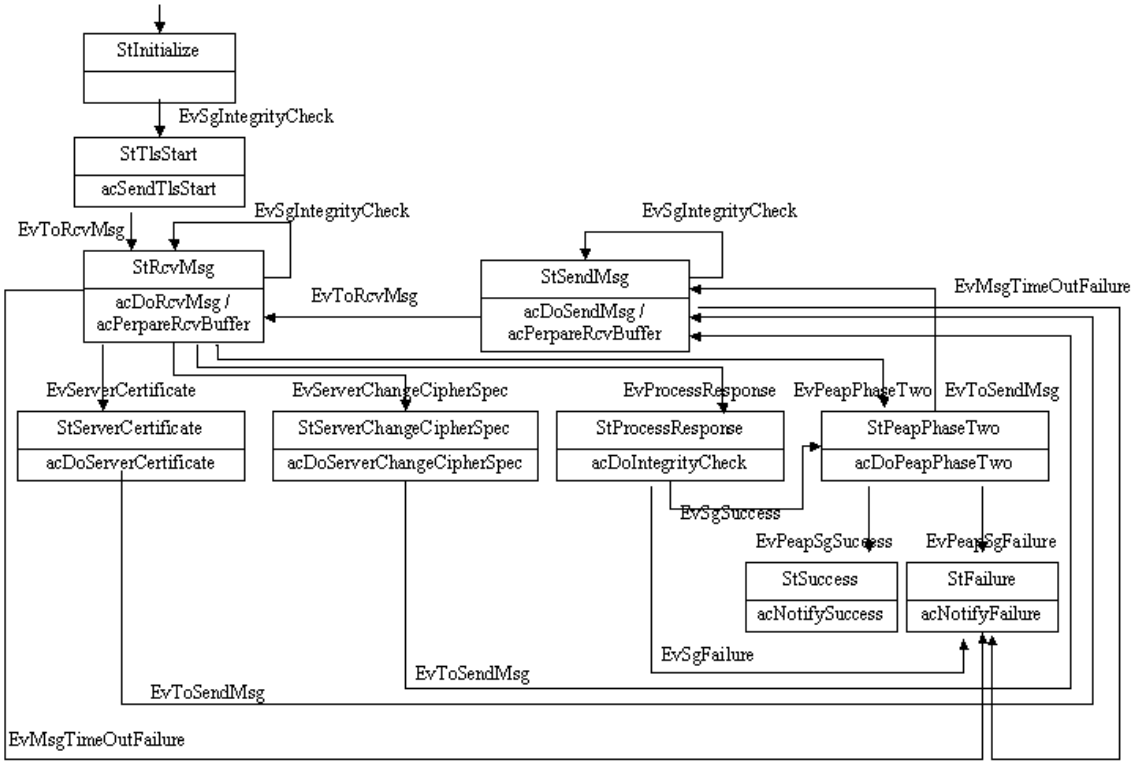

Figure 3.12: EAP-PEAP Method State Machine

In phase 2 - PEAP Tunnel Authentication StPeapPhaseTwo Do tunneled authentication, only support MS-Chap-V2.

StPeapSuccess A final state indicating this entire PEAP Authentication is success.

StPeapFailure A final state indicating this entire PEAP Authentication is failure.

- **Actions of EAP-PEAP Method State Machine**

  In phase 1 - TLS Handshake Protocol

  These also defined in Section 3.3.2. acSendTlsStart, acPrepareRcvBuffer, acDoRcvMsg, acDoSendMsg, acDoServerCertificate, acDoServerChangeCipherSpec, acDoIn-

tegrityCheck.

In phase 2 - PEAP Tunnel Authentication

acDoPeapPhaseTwo This action have two steps, Step1).Encrypt peap message, those will encapsulate of AVPs within peap message. Step2).Determine this authenticated method in the phase2 with receive message.

acNotifyPeapSuccess Update the policy element into PolicyOnSuccess, and generate EvSgEndMethod to notify upper layer.

acNotifyPeapFailure Update the policy element into PolicyOnFailure, and generate EvSgEndMethod to notify upper layer.

- **Events of EAP-PEAP Method State Machine**

  In phase 1 - TLS Handshake Protocol

  These also defined in Section 3.3.2. EvSgSuccess, EvSgFailure, EvToRcvMsg, EvToSendMsg, EvServerCertificate, EvServerChangeCipherSpec, EvProcessResponse, EvMsgTimeoutToFailure.

  In phase 2 - PEAP Tunnel Authentication

  EvPeapSgSuccess Set in StPeapPhaseTwo state. Let process into StPeapSuccess.

  EvPeapSgFailure Set in StPeapPhaseTwo state. Let process into StPeapFailure.

  EvPeapPhaseTwo Set in StProcessResponse state. Let process into EvPeapPhaseTwo.

  If MS_CHAPV2_SUPPORT_SUCCESS_STRING is TRUE, then ...

  EvPeapPhaseTwo Set in StProcessResponse or StRcvMsg state. Let process into EvPeapPhaseTwo or StSendMsg.

Chapter 4

Testbed

We have implemented a testbed for the WIRE Diameter. General IEEE 802.1x authentication diagram is Fig. 4.1. The concept of this architecture have three roles, they are *Client*, *Authenticator* and *Server* respectively. The authenticator usually is AP which is responsible to transmitting layer 2 frames and AAA messages in AAA architecture, even restrict network traffic with port-based policy. Unfortunately, these is still not any AP supports Diameter protocol. WIRE Diameter must also implement NAS functionality. AP in this testbed only play roles of a dumb bridge between wire and wireless network. The NAS will take responsible for most jobs of access control. According to these requests, a WIRE Diameter authentication diagram is Fig. 4.2 with an additional role, NAS. We present a testbed as Fig. 4.3. Diameter NAS must transform message between EAP-Packet and Diameter-Message, and blocking network traffic. So we can put actual AP and Diameter NAS into a same machine for being a Diameter AP. In this assuming case, the MN could successfully roaming from Diameter AP 1 to Diameter AP 2. After authenticated MN, Diameter NAS will open up the use of network for the MAC address of MN. Then MN can connect to Internet through Diameter NAS. In the following, We described fully authentication flow of WIRE Diameter. In Fig. 4.4, We illustrate MD5-CHALLENGE authentication in WIRE Diameter for understanding whole flows. The diagram can slice into two part, one part is typical 802.1x

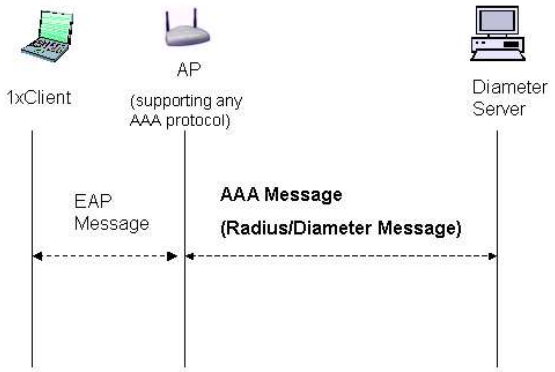

Figure 4.1: Typical 802.1x Architecture

authentication flows, another is WIRE Diameter authentication procedure in Fig. 2.9 (see Section 2.3.4).

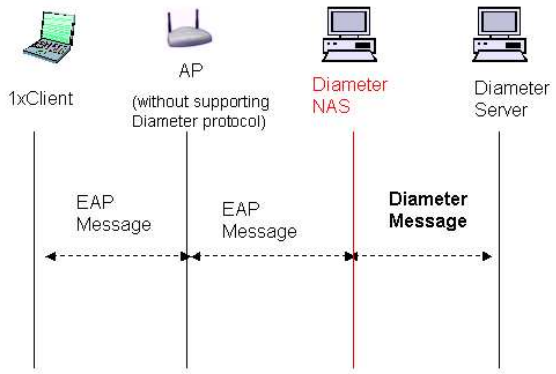

Figure 4.2: Diameter-unsupported AP in 802.1x Architecture

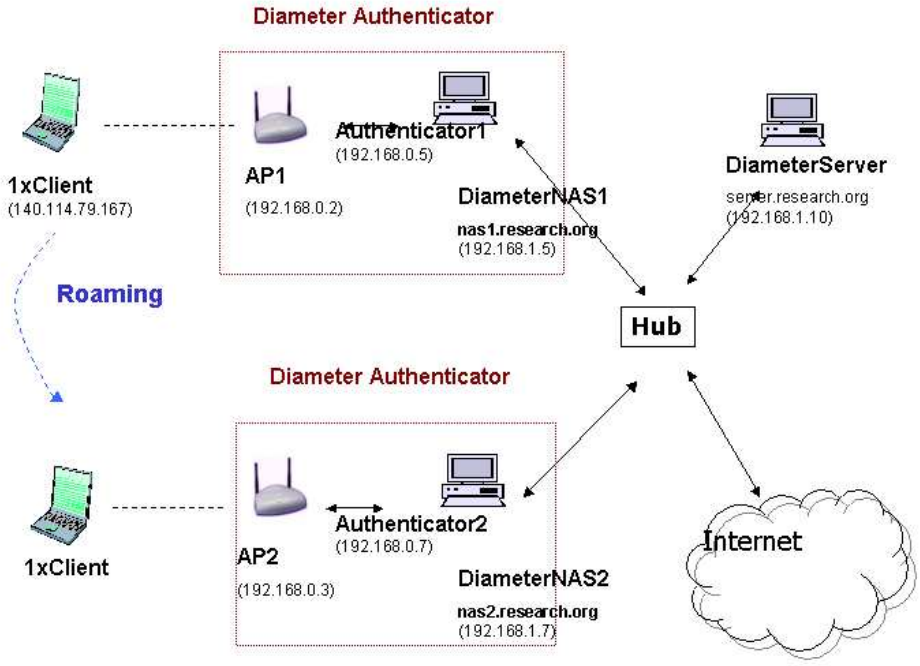

Figure 4.3: Testbed of WIRE Diameter

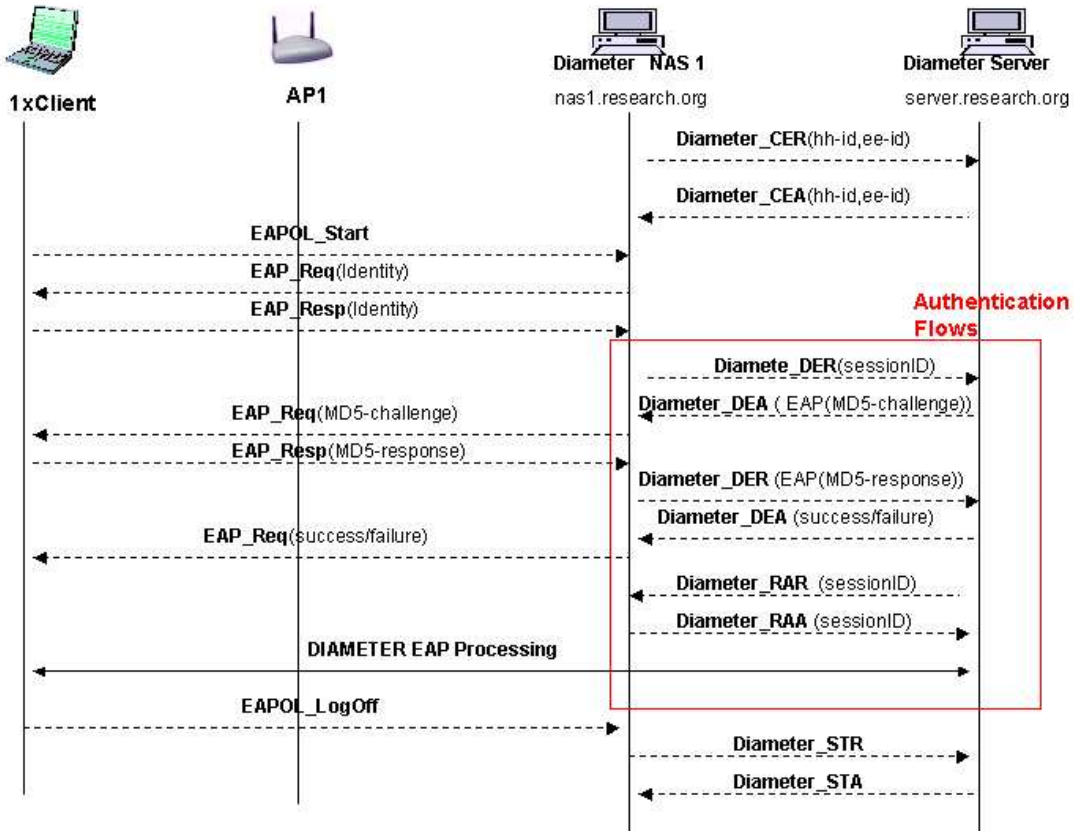

Figure 4.4: Fully Authentication Flow in WIRE Diameter

Chapter 5

Summary

The WIRE Diameter is a free AAA software conformed to Diameter base protocol and Diameter EAP Application. It supports various EAP authentication methods, including EAP-MD5, EAP-TLS, EAP-TTLS and PEAP. In the design and implementation, we pay special attention to platform independent. Therefore, the WIRE Diameter works on Linux, FreeBSD and various versions of MS Windows.

Although WIRE Diameter emphasize Authentication and Authorization; Accounting could also be easily extended in this project. Diameter server and connected Diameter NASs regularly communicates by Accounting-Message, server could store allied bill information. However, We have not intention to stock a great deal of knowledge currently.

Due to WIRE Diameter must work with Diameter NAS, the actual AP must be set transmit packet without handling any EAPOL frame. In this case, the software of client sends EAPOL-START with PAE group address (see [10]) for authenticating. We list some of famous softwares currently in table 5.1.

The WIRE Diameter could be a generic AAA server in any network environment. Particularly, we have implemented it as an AAA server over WLANs. It provides many advantages than RADIUS. Furthermore, we have also implemented a Diameter NAS to translate

EAP and Diameter messages over WLANs. By reading this paper, we hope people should be able to examine the source code of WIRE Diameter in addition to using it. We hope the release of WIRE Diameter would be useful to improve the security in wireless networks.

	Xsupplicant	Xsupplicant	Aegis	Aegis
Version	0.8	1.0	1.6.3	2.2.0
Supported Methods	MD5,TLS, TTLS,PEAP	MD5,TLS, TTLS,PEAP	MD5,TLS TTLS	MD5,TLS, TTLS,PEAP
Workable	No	Yes	Yes	No
Comment	Not stable	—	—	Not support PAE group address

Table 5.1: Compatible Client Softwares to WIRE Diameter

Bibliography

[1] "OpenDiameter." http://www.opendiameter.org.

[2] "Industrial Technology Research Institute (ITRI), Taiwan." http://www.itri.org.tw.

[3] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "Diameter Base Protocol," Sept. 2003.

[4] P. Eronen, T. Hiller, and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application." IETF Internet Draft, <draft-ietf-aaa-diameter-eap-08.txt>, work in progress, June 2004.

[5] "WIRE Diameter." http://wire.cs.nthu.edu.tw/wirediameter.

[6] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (RADIUS)." IETF RFC 2865, June 2000.

[7] "IETF AAA WG." http://www.ietf.org/html.charters/aaa-charter.html.

[8] "3GPP TS 29.229 – Cx and Dx interfaces based on the Diameter protocol,Version 6.0.0," May 2004.

[9] J. Loughney, "Diameter Command Codes for Third Generation Partnership Project (3GPP) Release 5." IETF RFC 3589, Sept. 2003.

[10] I. S. 802.1X-2001, "IEEE Standard for Local and metropolitan area networks- Port-Based Network Access Control," Oct. 2001.

[11] P. R. Calhoun, T. Johansson, C. E. Perkins, T. Hiller, and P. J. MaCann, "Diameter Mobile IPv4 Application." IETF Internet Draft, <draft-ietf-aaa-diameter-mobileip-18.txt>, work in progress, May 2004.

[12] B. Aboba, P. Calhoun, S. Glass, T. Hiller, P. McCann, H. Shiino, P. Walsh, G. Zorn, G. Dommety, C. Perkins, B. Patil, D. Mitton, S. Manning, M. Beadles, S. Sivalingham, A. Hameed, M. Munson, S. Jacobs, B. Lim, B. Hirschman, R. Hsu, H. Koo, M. Lipford, E. Campbell, Y. Xu, S. Baba, and E. Jaques, "Criteria for Evaluating AAA Protocols for Network Access." IETF RFC 2989, Nov. 2000.

[13] "FreeRadius." http://www.freeradius.org.

[14] "Co-Existence of RADIUS and Diameter," May 2003.

[15] B. Aboba and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile." IETF RFC 3539, June 2003.

[16] B. Aboba and P. Calhoun, "RADIUS Support For Extensible Authentication Protocol (EAP)." IETF RFC 3579, Sept. 2003.

[17] S. Kent and P. Atkinson, "Security Architecture for the Internet Protocol." IETF RFC 2401, Nov. 1998.

[18] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)." IETF RFC 2406, Nov. 1998.

[19] L. Ong and J. Yoakum, "An Introduction to the Stream Control Transmission Protocol (SCTP)." IETF RFC 3286, May 2002.

[20] B. Aboba, J. Arkko, and D. Harrington, "Introduction to Accounting Management." IETF RFC 2975, Oct. 2000.

[21] C. Rigney, "RADIUS Accounting." IETF RFC 2866, Oct. 2000.

[22] M. Chiba, M. Eklund, D. Mitton, and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)." IETF RFC 3576, July 2003.

[23] P. R. Calhoun, S. Farrell, and W. Bulley, "Diameter CMS Security Application." IETF Internet Draft, <draft-ietf-aaa-diameter-cms-sec-04.txt>, Mar. 2002.

[24] T. Dierks and C. Allen, "The TLS Protocol." IETF RFC 2246, Jan. 1999.

[25] S. Kent and R. Atkinson, "IP Authentication Header." IETF RFC 2402, Nov. 1998.

[26] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)." IETF RFC 2409, Nov. 1998.

[27] L. Blunk and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)." IETF RFC 2284, Mar. 1998.

[28] R. Rivest, "The MD5 Message-Digest Algorithm." IETF RFC 1321, Apr. 1992.

[29] P. Funk and S. Blake-Wilson, "EAP Tunneled TLS Authentication Protocol." IETF Internet Draft, <draft-ietf-pppext-eap-ttls-04.txt>, work in progress, Apr. 2004.

[30] H. Andersson, S. Josefsson, G. Zorn, D. Simon, and A. Palekar, "Protecting EAP Protocol (PEAP)." IETF Internet Draft, <draft-josefsson-pppext-eap-tls-eap-05.txt>, Sept. 2002.

[31] J. Vollbrecht, P. Eronen, N. Petroni, and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator." IETF Internet Draft, <draft-ietf-eap-statemachine-01.txt>, work in progress, June 2003.

[32] B. Lloyd and W. Simpson, "PPP Authentication Protocols." IETF RFC 1334, Oct. 1992.

[33] W. Simpson, "PPP Challenge Handshake Authentication Protocol(CHAP)." IETF RFC 1334, Aug. 1996.

[34] G. Zorn and S. Cobb, "Microsoft PPP CHAP Extensions." IETF RFC 2433, Oct. 1998.

[35] G. Zorn, "Microsoft PPP CHAP Extensions, Version 2." IETF RFC 2759, Jan. 2000.

[36] H. Andersson and S. Josefsson, "Protecting EAP with TLS (EAP-TLS-EAP)." IETF Internet Draft, <draft-josefsson-pppext-eap-tls-eap-00.txt>, Aug. 2001.

[37] A. Palekar, D. Simon, G. Zorn, J. Salowey, H. Zhou, and S. Josefsson, "Protecting EAP Protocol (PEAP), Version 2." IETF Internet Draft, <draft-josefsson-pppext-eap-tls-eap-07.txt>, Oct. 2003.

[38] "Adaptive Communication Environment(ACE)." http://www.cs.wustl.edu/ schmidt/ACE.html.

[39] "Xerces C++ Parser." http://xml.apache.org/xerces-c/index.html.

[40] "OpenSSL." http://www.openssl.org.

[41] "Pcap Library." http://www.tcpdump.org.

[42] "WinPcap Library." http://winpcap.polito.it.

[43] "Libnet Library." http://libnet.sourceforge.net.

[44] "IEEE-802-1X-REV, Draft Standard for Local and Metropolitan Ares Networks : Port-Based Network Access Control(Revision)," Jan. 2004.