freeDiameter: An Open Source Framework for an Authentication, Authorization, and Accounting Infrastructure

Sébastien Decugis  Fumio Teraoka

AAA (Authentication, Authorization, and Accounting) is one of the important functions indispensable for providing services on the Internet. Diameter Base Protocol was standardized in IETF as a successor of RADIUS, which is a widely used AAA protocol in the current Internet. Diameter solves the problems that RADIUS has such as support of multiple realms, reliable and secure message transport, and failover. There are several open source implementations of Diameter Base Protocol. However, none of them completely conforms to the specification. The first contribution of freeDiameter is that it is an open source of Diameter Base Protocol that completely conforms to the specification. It is written in C and based on a BSD-like license. In the Diameter architecture, a particular service on Diameter Base Protocol is defined as a Diameter application such as Diameter EAP application for WiFi network access control. The second contribution of freeDiameter is that the software architecture of freeDiameter makes it easy to implement Diameter applications as additional plug-ins. freeDiameter has already been distributed through our home page. freeDiameter with Diameter EAP application has been used in our laboratory for WiFi network access. It was also used for network control in the WIDE camp held in September 2010 for four days in which approximately 200 researchers attended. There was no problem on freeDiameter. This is good evidence of the stability of freeDiameter.

1 Introduction

AAA (Authentication, Authorization, and Accounting) is one of the important functions indispensable for providing services on the Internet. Authentication is a function that verifies the identity of a user or device that requests a service. Once this identity is confirmed, authorization takes place. Authorization is a function that verifies user’s rights to access the services. Afterwards, the service is granted to the authenticated user, and service usage information is recorded; this is accounting.

AAA protocols refer to a class of Internet protocols designed to carry the data necessary to authentication, authorization, and accounting. The most deployed AAA protocol in the Internet nowadays is RADIUS[16]. However, RADIUS and other alternatives were not designed to fit the requirements of the modern information society. The Internet is a network of networks; a lot of administrative domains called realms are interconnected. Therefore, the AAA protocol must support a multi-realm environment. To carry the AAA information in a multi-realm environment, the AAA protocol must have reliable and secure message transport mechanisms. To keep high availability of the AAA infrastructure even if some AAA nodes fail, the AAA protocol must have failover mechanisms. However, RADIUS does not have these functions.

IETF (Internet Engineering Task Force) has standardized Diameter Base Protocol[8] in 2003 as
a successor of RADIUS. Diameter Base Protocol is a protocol to carry the AAA information reliably and securely between the AAA client and the AAA server. A particular service on Diameter Base Protocol is defined as a Diameter application such as Diameter EAP application[11] which provides network access control. Diameter Base Protocol was designed from scratch with the focus on reliability, security, maintainability, multi-realm support, and extensibility. As a result, Diameter Base Protocol is very complex. This complexity makes the deployment of Diameter slow in the past decade.

Our project is dedicated to study the AAA infrastructure for New Generation Networks[18]. When the project started in 2008, there were some Diameter stacks such as OpenDiameter[3]. However, none of them satisfied the requirements for the AAA infrastructure, i.e., conformance with the specification, performance, and extensibility. For these reasons, the freeDiameter project started. In 2010, the first version of freeDiameter was released[2]. freeDiameter completely conforms to the specification. The software structure of freeDiameter makes it easy to implement Diameter applications on it. freeDiameter is written in C for performance and polyvalence that supports natively all the features of Diameter Base Protocol. Since the license of freeDiameter is based on BSD-like license, users can freely modify freeDiameter, distribute modified freeDiameter, and use it for any purpose.

This paper is organized as follows. Sec. 2 introduces an overview of the Diameter architecture. Sec. 3 surveys other open Diameter stacks. Sec. 4 describes the software architecture of freeDiameter. Sec. 5 shows the API of freeDiameter provided to Diameter applications and some examples of Diameter agents and a Diameter application. Sec. 6 shows the basic performance of freeDiameter and a result of an experiment in an actual environment which shows stability of freeDiameter. Sec. 7 concludes this paper.

2 Overview of Diameter Architecture

2.1 Diameter Nodes

Diameter defines several kinds of nodes. The Diameter client is a node that sends AAA requests. An example of the Diameter client is a NAS (Network Access Server). The Diameter server is a node that handles AAA requests for a particular realm. The relay agent relays the Diameter messages without analyzing their contents. The proxy agent relays the Diameter messages and it may modify the Diameter messages based on the policy of the realm. The redirect agent does not forward the Diameter messages but notifies the Diameter client, the relay agent, or the proxy agent of the routing information. The translation agent performs protocol translation between Diameter and other AAA protocols such as RADIUS.

Figure 1 shows an example of the Diameter nodes. There are two realms: Realm-1 and Realm-2. Each realm has its own Diameter server that handles AAA requests for the realm, the relay or proxy agent for message routing to other realms, the redirect agent, and the NAS (Network Access Server) as the Diameter client. The Diameter client is also the translation agent that performs protocol translation between Diameter and, e.g., RADIUS. In this example, a node node@realm-2 sends a network access request to the NAS in Realm-1. The Diameter Request and Answer messages are exchanged between the Diameter client in Realm-1 and the Diameter server in Realm-2 via the relay/proxy agents in Realm-1 and Realm-2.

2.2 Diameter Base Protocol and Diameter Applications

Diameter consists of Diameter Base Protocol [8][12] and a lot of Diameter applications such as Diameter EAP application[11] and Diameter SIP application[14] in IETF, and Diameter S6a/S6d application[5] in 3GPP. Diameter Base Protocol governs the management of the connections between the Diameter nodes. When a connection between
two nodes is established, they negotiate their capabilities by exchanging the messages. Afterwards, the connection is actively maintained with periodic exchange of the watchdog messages.

Diameter Base Protocol runs over TCP or SCTP (Stream Control Transmission Protocol) [17] for reliability and mandates the use of IPsec or TLS for security. It also defines the routing mechanism of Diameter messages, as well as the behavior in case of disconnection of a remote Diameter node. In other words, at the Diameter Base Protocol level, all the Diameter nodes are forming an overlay peer-to-peer network with specific security, reliability, and routing properties.

On top of this overlay network, Diameter applications are deployed. A Diameter application specifies the role of the different entities involved in a particular service, and the different commands and data that these entities exchange. For example, Diameter EAP application defines some commands and data for authentication and authorization of a node that tries to connect to a network.

2.3 Message Format and Dictionary

The Diameter messages consist of the header followed by any number of the AVPs (Attribute Value Pairs) as shown in Figure 2. A Diameter exchange always starts with the Request message followed by the Answer message. The message header contains a flag ‘R’ that differentiates the requests from the answers. It also contains a command code that identifies the type of the message. For example, the messages involved in connection management have the application identifier 0 (Base Protocol) and the codes 257 for Capability-Exchange, 280 for Device-Watchdog, and 282 for Disconnect-Peer messages.

Each AVP starts with a code that identifies its contents, followed by a flag field, a length, an optional vendor identifier, and the actual data. Some attributes are used to carry the data that belongs to Diameter Base Protocol in commands that pertain to an application. For example, the Destination-Realm AVP (code 283) identifies the destination domain of a message, and the Route-Record AVP (code 282) is added by each Diameter node that forwards a message and is used to detect routing loops as well as provide message traceability.

Because the data type contained in an AVP is not indicated in the AVP header, each Diameter node must maintain a mapping between the AVP codes and the data types, called a dictionary. Each Diameter command has a number of AVPs that must be present in this command, as well as optional AVPs. The grammar of each command attributes is defined in the Augmented Backus-Naur Form (ABNF) [9]. Except for a few exceptions, the commands are allowed to contain additional AVPs that are not specified in their ABNF definition. This allows extending the usage of some commands or reusing the commands in new applications more easily.

2.4 Message Routing

Routing the Diameter Requests is performed as follows. There are several elements that must be considered: the Destination-Realm AVP in the message that indicates the target domain, the application identifier of the message so that it is forwarded to a node that supports the application, and also the local routing table and the status of the different connections. Once a next hop has been selected, the Diameter node adds a Route-Record AVP that contains the identity of the previous Diameter node that sent this message, then sets a new hop-by-hop identifier and sends the message. Routing the Diameter Answers is simple. The Answer message always follows the same path as the Request backward.

A local copy of the sent Requests must be kept, and if the connection is torn down, this copy is sent again through another connection, or an error
Table 1 Comparison of Diameter implementations

<table>
<thead>
<tr>
<th>Product</th>
<th>License</th>
<th>Language</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenDiameter</td>
<td>LGPL, GPL</td>
<td>C++</td>
<td>Not maintained anymore, cannot be compiled on recent systems.</td>
</tr>
<tr>
<td>OpenBlox (Traffix Systems)</td>
<td>Non standard</td>
<td>C++, Java</td>
<td>Limited features in free version, no routing capabilities, no SCTP or TLS support, focus on IMS interfaces.</td>
</tr>
<tr>
<td>Codename Circumference</td>
<td>LGPL</td>
<td>C, C++</td>
<td>Active development, no routing capabilities, no support of Diameter agents, focus on WebAuth application.</td>
</tr>
<tr>
<td>freeDiameter</td>
<td>BSD</td>
<td>C</td>
<td>Active development, completely conform to the specification, general-purpose Diameter framework.</td>
</tr>
</tbody>
</table>

is generated if the delivery is not possible. This mechanism, called failover, ensures that all Diameter Requests receive a Diameter Answer, which can be an error. This provides reliability at the application layer, which is important especially for accounting.

2.5 Message Transport
SCTP (Stream Control Transmission Protocol) [17] is the recommended transport protocol for Diameter, benefiting from the multi-streams or unordered delivery capabilities which can reduce the head-of-the-line blocking compared to TCP as RFC 3539[6] highlights. AAA protocols convey sensible information both in terms of security and privacy of the user. Diameter Base Protocol establishes a hop-by-hop secure channel between Diameter nodes either with IPsec[15] or TLS[10]. There is no end-to-end protection available. This is partly due to the fact that in most cases one or more AAA proxies are located on the path of a message, and these proxies must be able to access and change the content of the message.

3 Related Work
There are several Diameter stacks available for free in October 2010, omitting the variants based on the same stack. Table 1 shows the comparison of freeDiameter and other stacks.

3.1 OpenDiameter
The OpenDiameter[3] implementation was developed mainly as a proof of the concept at the same time when the Diameter protocol was first specified in IETF. This implementation relies heavily on other libraries for core features: Xerces for XML parsing, ACE for communications, Boost for parsers, and OpenSSL for cryptography. Because the programming interface of some of these dependencies has changed since the last release of OpenDiameter in March 2007, this software cannot be compiled anymore without performing a lot of modifications. The OpenDiameter implementation consists of a set of libraries written in C++. The Diameter Base protocol support is not up to date with the Diameter specification.

3.2 OpenBlox
In September 2010, Traffix Systems[4] has released an open source version of its Diameter stack called OpenBlox, available in Java and C++. This open source version has limited features: no SCTP or TLS support, no support for the latest Diameter protocol evolutions, no routing capabilities. In addition, the license of the open source version restricts its use to development and test purpose only.

3.3 Codename Circumference
The Codename Circumference (Circum)[1] project started at about the same time as freeDiameter. Circumference is written in C, and a C++ wrapper is provided. It implements the Diameter specification up to the latest updates. It was designed and developed to accommodate a particular Diameter application, WebAuth, with a simple client / server structure. As a result, the implementation might
not be suitable to implement Diameter agents that relay messages, because it does not deal with routing issues, or complex servers, because of limited support in this area.

4 Software Architecture of freeDiameter

This section shows how freeDiameter was designed and implemented to leverage the Diameter architecture and provide a new reference stack for Diameter Base Protocol.

4.1 Software Structure

There are two possible structures to implement Diameter Base Protocol and Diameter applications. The application programming interface to Diameter Base Protocol discussed in IETF assumed the structure shown in Figure 3-(a), in which Diameter Base Protocol is handled in the library and a Diameter application is executed in the main module. However, this structure makes it difficult to implement several Diameter applications on a single Diameter node. Therefore, freeDiameter takes another approach shown in Figure 3-(b). In freeDiameter, the main module handles the Diameter Base Protocol tasks, i.e., management of the connections and routing while the extensions loaded as plug-ins add the logic of each Diameter application. This design allows an easy deployment of several applications on a single Diameter node, e.g., authentication and authorization servers. Thus, it is possible to centralize all Diameter services on a single Diameter node at the beginning, and later, to move some services to different Diameter nodes when the load increases, without any code rewriting.

From source code viewpoint, freeDiameter consists of the main module, the library, and optional extensions. The library contains the functions necessary to handle the Diameter protocol objects: the dictionary, the messages, and the AVPs. It also provides the functions to manipulate the Diameter session states, the queues, and a few other facilities such as debugging. This library is used by both the main module and the extensions.

In execution state, freeDiameter is a daemon running in the user space of the operating system. The freeDiameter daemon manages the connections to other Diameter nodes, according to the connection state machine defined in Diameter Base Protocol. The core of the daemon consists of three message queues: – incoming, local, and outgoing – and three classes of threads – Routing-IN, Routing-OUT, and Dispatch. In order to cope with variable load, several instances in each class of threads can be dynamically created if needed.

4.2 Message Handling

The message flow is depicted in Figure 4. Messages received from the network are stored in the incoming queue after initial parsing and validation (Fig. 4-(1)). Threads in the Routing-IN class pick the items from this queue (Fig. 4-(2)) and decide if they should be handled locally or forwarded to another Diameter node, according to the message
contents and the locally registered extensions, then requeue them in the local or outgoing queue accordingly (Fig. 4-(3)). Other threads from the Dispatch class handle the messages picked from the local queue to the registered extensions callbacks after dictionary validation (Fig. 4-(4)). These registered callbacks, which implement Diameter applications logic, process the messages and generate the Answers to the Requests. The extensions may also at any time create new Requests and send them. The messages to be sent are stored in the outgoing queue (Fig. 4-(5)), where Routing-OUT threads pick them (Fig. 4-(6)) and decide towards which Diameter node they will be sent next (Fig. 4-(7)).

When a Diameter Request is sent, a copy is kept until the corresponding Answer is received. In case that the connection is torn down before this Answer arrives, the message is simply requeued to be sent to a different Diameter node (Fig. 4-(8)). This provides the failover mechanism described in Sec. 2.4.

### 4.3 Extension Handling

The extensions are loaded when the daemon starts. An initialization routine is called by the daemon and performs all callback registrations. After this initial setup, the daemon behavior is event-driven; actions are triggered by incoming messages, network connections, or events generated in the extensions. The current freeDiameter repository contains a few extensions, e.g., a Diameter EAP server, a RADIUS/Diameter translation gateway, a Diameter SIP server, etc. Some extensions only add new object definitions in the dictionary; keeping dictionary definitions separate from applications allows easy reuse and a useful feature since many Diameter applications use the attributes defined in other applications.

There are hooks in the freeDiameter framework that allow extensions to modify the behavior of all threads. For example, an extension can implement advanced Diameter routing behavior by registering a callback in the Routing-OUT hook that will influence the message routing process. This powerful yet simple mechanism works as follows: when a message is ready to be sent, a list of all Diameter nodes with an open connection is built. All Diameter nodes that appear in the list of the Route-Record AVPs in the message are removed from the list, as well as all Diameter nodes from which an error message has been received previously for this message. The result is a list of potential candidate Diameter nodes. An initial number of points is given to each Diameter node depending on some criteria such as: “did the Diameter node advertise the message’s application?” and “does the realm advertised by the Diameter node match the message Destination-Realm value?” Afterwards, this list is passed to all registered callbacks, which can change the score of each candidate. For example, if a known user must be directed to a special different server, it is possible to do so very easily here. Once all callbacks have processed the list, the message is sent to the Diameter node with the highest number of points. The list is kept in cache with the message, so that if an error is received or if the connection is torn down and failover occurs, the message is sent again to the second Diameter node in the list, and so on. When no Diameter node remains in the list with a positive mark, an error is generated and returned as the Answer.

### 4.4 Dictionary Structure

As mentioned in Sec. 2.3, supporting the Diameter dictionary is the first step required to parse a Diameter message. Most implementations allow the dictionary definitions to be loaded from an XML file [13]. This approach allows changes in the dictionary definitions without requiring changes in the software. However, freeDiameter takes a different approach. The dictionary definitions are specified in C language. Although this offers less flexibility, it allows more features than the limited XML definitions do. For example, in freeDiameter, it is possible to specify ABNF rules for the Grouped AVP objects, i.e., the AVPs that contain other AVPs. It is also possible to specify special formatting callbacks, e.g., to display more nicely some binary information such as IP addresses. It is still possible to create an XML parser module that will read a dictionary file in the legacy format and add the corresponding definitions through the programming interface.

The dictionary module is implemented as hierarchical chained lists, indexed on different values, in most cases, the name and the code of the objects. The available objects are vendors, applications, types, constants, AVP, commands, and rules.

Figure 5 shows an example of the hierarchical list.
Fig. 5 An example of dictionary structure

structure of the dictionary, in this case for the vendors, AVPs, and rules hierarchy. In this figure, the top-level list is the list of vendors ordered by their identifier value. Each vendor item contains a list of the AVPs that this vendor defined. The standard AVPs defined in IETF are attached to vendor with identifier 0. Then, for AVPs in the list that have the Grouped AVP type, lists of ABNF rules are attached.

The complete dictionary consists of several such hierarchical lists for different objects. This hierarchical structure was designed to accelerate the parsing of the messages. For example, when an AVP is being parsed, the AVP code and vendor are available in the AVP header. The search function first resolves the vendor, then search for the AVP code in the AVP sublist for this vendor. Once the AVP is found, the list of rules that must be verified is directly available.

4.5 AVP Handling

When a message is received, it is parsed a first time to validate its format. During this process, the boundaries of each AVP is checked based on their length field and padding algorithm, and the AVP objects are allocated and initialized with pointers to the AVP data in the message. During this pass, no dictionary lookup is performed. If the message is forwarded to another Diameter node, a new Route-Record AVP is added in the list and the resulting Diameter message is sent without further parsing.

When the message is to be handled locally, a second parsing is performed to check the message conformance. During this second pass, each AVP is looked up in the dictionary, and if a definition is found, the attribute value is interpreted accordingly, i.e., integers and floats are converted from network byte order, grouped AVPs are parsed and children AVP created, and octet strings values are copied verbatim, so that when this pass is completed the original message buffer can be freed to save memory.

Conformance to the ABNF rules is also verified. For each object with a children list, freeDiameter checks in the dictionary if there are rules defined for this object. If such rules are found, they are verified one by one by counting the relevant AVP occurrences in the children list.

Once the message has completed the dictionary validation, it is passed to upper layers in the implementation. Therefore, the application does not need to handle message validation because it is already performed in the main module. The same validation is performed on outgoing messages, to avoid sending invalid Diameter messages in the network and more quickly find the source of an error.

4.6 Transport Abstraction Layer

The details of the transport layer in freeDiameter are hidden in a wrapper called cnxctx. This wrapper provides an unified abstraction to the remaining of the daemon, while taking care of the details such as SCTP streams or TLS protection. SCTP is natively supported in freeDiameter but can be disabled at compilation if the underlying operating system does not support this feature. TLS support is provided by the GnuTLS library.

To incite users to actually protect Diameter communications with TLS, the freeDiameter daemon will refuse to start if no valid certificate is configured. TLS can then be disabled for each connection as needed. This opt-out approach for TLS will result in a better security of the AAA deployments. The reason to encourage TLS protection instead of IPsec is that it allows the use of separate security credentials (certificate, authority) for the Diameter

† GnuTLS is an open-source library that provides support for TLS 1.0, TLS 1.1, TLS 1.2 and SSL 3.0 protocols. http://www.gnutls.org
subsystem, allowing a separation from other applications which might be compromised.

5 Applications API and Extension Examples

As mentioned in previous sections, the freeDiameter daemon provides the support of the Diameter Base Protocol, which is used to establish and maintain an overlay network between Diameter nodes. Extensions can add the handling of Diameter application messages creating a Diameter client, server, or agents. Extensions may also add special routing behaviors to implement an intelligent Diameter routing agent. This section shows how easy it is to deploy a Diameter application using the freeDiameter framework by giving a few extensions examples.

5.1 freeDiameter framework API

freeDiameter extensions can use both the functions from the freeDiameter library and the main module. The available functions form the Application Programming Interface (API) of the freeDiameter framework. The library functions are categorized into eight modules: debug, dictionary, sessions, messages, dispatch, queues, peers, and routing. These modules are easily recognizable by their naming scheme, i.e., the logging functions are prefixed with `fd_log`, the dictionary functions with `fd_dict`, and so on. Table 2 shows the main API functions (not exhaustive).

5.2 RADIUS/Diameter gateway

The `app_radgw.fdx` extension implements a gateway that translates between RADIUS and Diameter. This extension is a client from the Diameter viewpoint and a server from the RADIUS viewpoint. It supports conversion to several Diameter applications, which are provided by plug-ins. Currently, conversion to Diameter EAP application, Diameter SIP application, and Diameter Base Accounting are supported.

This extension does not register any callback on the dispatching thread, because it does not act as a Diameter server. Upon initialization, it only adds the supported applications to the list advertised by the daemon in the capability exchange by calling `fd DISP_APP_SUPPORT()`, and opens two RADIUS server sockets for authentication/authorization and accounting. When a new RADIUS message is received, it is parsed and a new Diameter message is created, using the functions from the library (`fd_msg_new()`, `fd_msg_avp_new()`, `fd_msg_avp_add()`). The created Diameter Request is sent using the interface of the mail modules `fd_msg_send()`, registering at the same time a callback to be called when the matching Answer is received. The extension does not need to deal with routing the message, the main module will handle this part from the message content and the existing connections. When the Diameter Request reaches a server such as a Diameter EAP server (see Sec. 5.3), an Answer is generated.

When this Answer reaches the node that created the Request, the callback registered when the message was sent is called with the Answer received as parameter. In case of `app_radgw.fdx`, this callback creates a new RADIUS message and converts all attributes or error codes from the Diameter format to the RADIUS format, then sends the resulting message to the RADIUS client who sent the initial request.

To summarize, this typical Diameter client extension uses the message manipulation functions to create a new Diameter Request, and the `fd_msg_send()` function to send this Request message to a Diameter server and register a callback to be called when the corresponding Answer is received.

5.3 Diameter EAP server

The Diameter EAP server implementation[7] called `app_diamap.fd`† provides a full-featured authentication and authorization server based on the specification of Diameter EAP application[11], with support for loadable EAP methods and configurable authorization attributes.

In this case, the extension mainly uses the dispatching module and the session state support from the main module, in addition to the message handling functions from the library. When the daemon starts and loads `app_diamap.fd`, this extension does its initialization including parsing its configuration file, connecting to the users database, and loading its EAP methods plug-ins. Then, it registers a callback in the dispatching module.

†2 http://diamap.yagami.freediameter.net/
## Table 2 Main API functions

<table>
<thead>
<tr>
<th>Module</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>fd_log_debug()</td>
<td>Send a message to configured output.</td>
</tr>
<tr>
<td></td>
<td>TRACE_DEBUG()</td>
<td>Display log messages with context (thread, function, ...)</td>
</tr>
<tr>
<td>Dictionary</td>
<td>fd_dict_new()</td>
<td>Create a new object in the dictionary.</td>
</tr>
<tr>
<td></td>
<td>fd_dict_search()</td>
<td>Find an object in the dictionary.</td>
</tr>
<tr>
<td>Sessions</td>
<td>fd_sess_handler_create()</td>
<td>Register a handler to store states into.</td>
</tr>
<tr>
<td></td>
<td>fd_sess_new()</td>
<td>Create a new session identifier.</td>
</tr>
<tr>
<td></td>
<td>fd_sess_settimeout()</td>
<td>Set a lifetime for the data registered with a session.</td>
</tr>
<tr>
<td></td>
<td>fd_sess_state_store()</td>
<td>Associate opaque data with a session and handler.</td>
</tr>
<tr>
<td></td>
<td>fd_sess_state_retrieve()</td>
<td>Retrieve the opaque data, if any.</td>
</tr>
<tr>
<td>Messages</td>
<td>fd_msg_avp_new()</td>
<td>Create a new AVP.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_new()</td>
<td>Create a new command header.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_browse()</td>
<td>Explore a message tree of AVPs.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_add()</td>
<td>Add an AVP into a tree.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_search_avp()</td>
<td>Search for an AVP in a tree.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_avp_setvalue()</td>
<td>Set the attribute value.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_bufferize()</td>
<td>Create a buffer to send on the network.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_parse_buffer()</td>
<td>First level of parsing, create pointers to each AVP.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_parse_dict()</td>
<td>Second level, resolve AVP definitions in the dictionary.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_parse_rules()</td>
<td>Third level, check conformance to the rules.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_send()</td>
<td>Route and send a message.</td>
</tr>
<tr>
<td></td>
<td>fd_msg_new_answer_from_req()</td>
<td>Create an Answer header with preset data.</td>
</tr>
<tr>
<td>Dispatch</td>
<td>fd_disp_register()</td>
<td>Register a callback with a criteria (application, command...)</td>
</tr>
<tr>
<td></td>
<td>fd_disp_app_support()</td>
<td>Advertise an application in CER/CEA exchanges.</td>
</tr>
<tr>
<td>Queues</td>
<td>fd_fifo_post()</td>
<td>Add an element at the end of the queue.</td>
</tr>
<tr>
<td></td>
<td>fd_fifo_get()</td>
<td>Retrieve the first element of the queue, or wait until such element is available (blocking).</td>
</tr>
<tr>
<td></td>
<td>fd_fifo_tryget()</td>
<td>Retrieve the first element of the queue, or return an error (non-blocking).</td>
</tr>
<tr>
<td></td>
<td>fd_fifo_timedget()</td>
<td>Same as fd_fifo_get, but wait only for defined time.</td>
</tr>
<tr>
<td></td>
<td>fd_fifo_setthrhd()</td>
<td>Register a callback to be called when the number of items in the queue reaches a threshold.</td>
</tr>
<tr>
<td>Peers</td>
<td>fd_peer_add()</td>
<td>Add a new peer connection information.</td>
</tr>
<tr>
<td></td>
<td>fd_peer_validate_register()</td>
<td>Register a callback on incoming connections.</td>
</tr>
<tr>
<td>Routing</td>
<td>fd_rt_fwd_register()</td>
<td>Register a callback on forwarded messages.</td>
</tr>
<tr>
<td></td>
<td>fd_rt_out_register()</td>
<td>Register a callback on outgoing messages.</td>
</tr>
</tbody>
</table>

When a Diameter-EAP-Request is received with a Destination-Realm AVP value matching the local domain, this message is passed by the daemon to the extension callback. This callback uses a few attributes such as the User-Name AVP and the EAP-Payload AVP to generate the first Answer, calling `fd_msg_new_answer_from_req()`, then sends this Answer (`fd_msg_send()`) and saves the current EAP state machine and other related information along the session identifier and the handler that was created during initialization (`fd_sess_state_store()`). This Answer is routed to the Diameter client that issued the Request, which can be an EAP authenticator or a translation gateway. The EAP message will ultimately reach the user, who will create a new EAP mes-
sage and send it. The new message is encapsulated in Diameter in the client who reuses the same session identifier as the previous message, and sends toward the server that runs app_diamap.fdx.

When the server receives the new Diameter-EAP-Request with the same session identifier, it retrieves the state saved previously for this user by calling fd_sess_state_retrieve(), and advances the EAP state machine accordingly, generates the new Answer, and so on, until a final authentication and authorization decision can be reached, at which point the last Diameter-EAP-Answer is sent for this session, and when it reaches the client, the service is provided to the user and accounting starts.

Thus, in the case of a server, the dispatching module is the main part of the API that the extension uses, and it can also benefit from the support of the session state management to avoid storing and matching this information for each message received in the extension itself.

5.4 Routing agent

The rt_reg.fdx extension provides an advanced Diameter routing behavior based on regular expressions matching. This extension is not related to any Diameter application, and can be used independently on a Diameter client or a Diameter agent such as routing relay or proxy.

Upon initialization, this extension parses the configuration file and reads the AVP name and the list of regular expressions associated with Diameter node identities and scores. It then registers with fd_rt_out_register() a callback for all outgoing messages in the routing module and waits. When a new Request message is processed by the daemon and ready to be emitted, the registered callback receives the list of candidate Diameter nodes and the Request to be sent. It checks if the message contains the AVP declared in the configuration, and if it is found it examines the list of candidate Diameter nodes for the message. When a candidate Diameter nodes was also declared in the extension’s configuration, the AVP value is checked against the regular expression, and if it matches, the configured score is added to this candidate. Although this mechanism is simple, it allows very flexible routing rules to be established, e.g., “all messages for the domains A, B, and C are sent to our first broker’s proxy”.

This simple extension uses only the message manipulation functions to search the AVP (fd_msg_search_avp()), and the routing hook to register its callback (fd_rt_out_register()).

6 Evaluation

This section shows the basic performance of freeDiameter by a test application and its stability through an experiment of network access control in an actual environment.

6.1 Basic Performance

In order to evaluate the basic performance of freeDiameter, a very simple Diameter application similar to ping was developed. The client generates a simple message with a random value and sends it to the server and the server answers including the same value. This test application generates Diameter messages of 232 bytes length in average between Request and Answer. The throughput of freeDiameter was measured by sending a number of concurrent Requests first, and then new Requests when an Answer has been received so that the number of message flows is constant. The rate of incoming Answers was measured by counting the number of messages received during one minute. It is important to note that during all these tests, no message has been lost and no erroneous random value was received, which shows that the framework is reliable, even under stress condition.

In these test, two kinds of devices were used. One is a D-Link DIR-330 wireless access point running freeDiameter over OpenWRT†3. Another is a Dell Inspiron 530S computer running the Ubuntu system. In both cases, the GnuTLS library is in version 2.8.6, with default algorithms parameters, where the negotiated cipher suite is RSA_AES_128_CBC_SHA1. In this measurement, the freeDiameter daemon receives the Request message, optionally deciphers it when TLS is used, then parses against the dictionary and ABNF rules, creates a corresponding answer, validates this answer, optionally ciphers it, and finally sends it back. The bit rate was directly measured on the wire with a packet capture tool, wireshark. The link is

†3 OpenWRT is A GNU/Linux based firmware program for embedded devices such as residential gateways and routers. http://openwrt.org
Table 3 freeDiameter throughput in messages per second (msgps) and corresponding bit rate

<table>
<thead>
<tr>
<th>Device</th>
<th>CPU</th>
<th>clock</th>
<th>memory</th>
<th>TLS</th>
<th>Throughput</th>
<th>On wire</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR-330 (access point)</td>
<td>Broadcom BCM5836P</td>
<td>264MHz</td>
<td>32MB</td>
<td>Yes</td>
<td>158 msgps</td>
<td>0.96MBps</td>
</tr>
<tr>
<td>530S (computer)</td>
<td>Intel Core2 Duo</td>
<td>2.33GHz</td>
<td>2GB</td>
<td>Yes</td>
<td>3,790 msgps</td>
<td>23.4MBps</td>
</tr>
</tbody>
</table>

100MBps Ethernet. Table 3 shows the parameters and the results.

As shown in Table 3, TLS protection has a negative impact on performance and bandwidth consumption. The results also show that a standard desktop computer running freeDiameter is capable of handling more than 3,000 messages per second (actually almost 5,000 when debugging is disabled), which is more than enough for a small Diameter environment such as users authentication in a small to medium company. Larger Diameter deployments such as phone operators with many subscribers would require more powerful equipments, or using redundancy and load balancing of messages to handle a larger load.

6.2 Stability

Nowadays, operating systems running on mobile devices support the WPA2 Enterprise mode for WiFi access control. The WPA2 Enterprise mode uses IEEE802.1X for authentication and authorization. Usually, an IEEE802.1X message is converted to a RADIUS message by a WiFi access point and is forwarded to a RADIUS server, which processes authentication and authorization as shown in Figure 6-(a). If a RADIUS/Diameter gateway described in Sec. 5.2 is installed at the position of the RADIUS server, a RADIUS message is converted to a Diameter message and is forwarded to a Diameter EAP server described in Sec. 5.3 which processes authentication and authorization as shown in Figure 6-(b). Thus, by employing the RADIUS/Diameter gateway and the Diameter EAP Server, a mobile device such as a note-PC can benefit from the advantages of Diameter such as multi-realm support, reliable and secure message transport, and high availability without any modifications to the software on the mobile device.

To evaluate stability of freeDiameter, a system described above was built in the WIDE Project Camp held for four days in September 2010 in which approximately 200 people attended. In the venue, there were several WiFi access points which were controlled by a WiFi controller. There were two sets of the RADIUS/Diameter gateway and the Diameter EAP server. The reason why there are two sets of them is to virtually create two realms and to distribute the load. The system was able to provide a smooth wireless environment for the attendees who did not report any authentication problem. A total of 14,265 successful authentications were performed by the two Diameter EAP servers, adding stress a little by reducing the re-authentication time to 10 minutes in the third day of the venue. The accounting experiment was also successful. These results show that freeDiameter is stable enough for actual use.

7 Conclusion

The Diameter protocol was designed to, and is becoming, a common protocol for all entities that belong to the control-plane in the network, especially where this control plane is not separated from the data plane. This is achieved by the unique properties of Diameter protocol in terms of routing, reliability, and extensibility. This architecture allows easy interconnection of control-plane entities.
that can create new services and also accelerate the deployment of services in networks.

The freeDiameter framework implements these unique properties and gives an easy access to developers for implementing applications on top of this framework, through a powerful yet simple API. Implementing a Diameter client, server, or relay is a simple task with this framework, while the implementation design choices of the framework ensure a good performance and robustness. The stress tests as well as real-life daily use show that freeDiameter is reliable and does what is expected, while at the same time it allows research and experiments on new Diameter applications and usages.

Acknowledgements

The freeDiameter framework development is funded by the National Institute of Information and Communications Technology (NICT), within the frame of the AKARI New Generation Network Project[18]. Part of the code presented here was contributed by students of Teraoka Laboratory in Keio University, in particular the Diameter EAP and Diameter SIP server extensions.

We appreciate the cooperation of the network PC members of the WIDE Camp who provided the network infrastructure for the test operation of freeDiameter and the cooperation of the WIDE members who joined the test operation.

References