



Diameter

Copyright © 2011-2018 Ericsson AB. All Rights Reserved.
Diameter 1.12.2
April 25, 2018

Copyright © 2011-2018 Ericsson AB. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0> Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Ericsson AB. All Rights Reserved..

April 25, 2018



1 Diameter Users Guide

The diameter application is a framework for building applications on top of the Diameter protocol.

1.1 Introduction

The diameter application is an implementation of the Diameter protocol as defined by RFC 6733. It supports arbitrary Diameter applications by way of a **dictionary** interface that allows messages and AVPs to be defined and input into diameter as configuration. It has support for all roles defined in the RFC: client, server and agent. This chapter provides a short overview of the application.

A Diameter node is implemented by configuring a **service** and one or more **transports** using the interface module *diameter*. The service configuration defines the Diameter applications to be supported by the node and, typically, the capabilities that it should send to remote peers at capabilities exchange upon the establishment of transport connections. A transport is configured on a service and provides protocol-specific send/receive functionality by way of a transport interface defined by diameter and implemented by a transport module. The diameter application provides two transport modules: *diameter_tcp* and *diameter_sctp* for transport over TCP (using *gen_tcp*) and SCTP (using *gen_sctp*) respectively. Other transports can be provided by any module that implements diameter's *transport interface*.

While a service typically implements a single Diameter node (as identified by an Origin-Host AVP), transports can themselves be associated with capabilities AVPs so that a single service can be used to implement more than one Diameter node.

Each Diameter application defined on a service is configured with a callback module that implements the *application interface* through which diameter communicates the connectivity of remote peers, requests peer selection for outgoing requests, and communicates the reception of incoming Diameter request and answer messages. An application using diameter implements these application callback modules to provide the functionality of the Diameter node(s) it implements.

Each Diameter application is also configured with a dictionary module that provide encode/decode functionality for outgoing/incoming Diameter messages belonging to the application. A dictionary module is generated from a *dictionary file* using the *diameterc* utility. Dictionaries for the RFC 6733 Diameter Common Messages, Base Accounting and Relay applications are provided with the diameter application.

1.2 Usage

To be written.

1.3 Examples

Example code can be found in the diameter application's `examples` subdirectory.

1.4 Standards Compliance

Known points of questionable or non-compliance.

1.4.1 RFC 6733

- There is no support for DTLS over SCTP.

- There is no explicit support for peer discovery (section 5.2). It can possibly be implemented on top of diameter as is but this is probably something that diameter should do.
- The peer state machine's election process (section 5.6.4) isn't implemented as specified since it assumes knowledge of a peer's Origin-Host before sending it a CER. (The identity becoming known upon reception of CEA.) The possibility of configuring the peer's Origin-Host could be added, along with handling of the case that it sends something else, but for many applications this will just be unnecessary configuration of a value that it has no control over.

Commentary

A more detailed commentary on RFC 6733 follows. Its purpose is to (hopefully) clarify not only what is supported but how, given that semantics and features discussed in the RFC are not solely the responsibility of the diameter application: in many cases much depends on the configuration a user passes to diameter, the implementation of *diameter_app(3)* callback modules in particular.

Comments apply to all text following the preceding comment. Be sure to distinguish between capitalized **Diameter**, the protocol defined by the RFC, and lowercase **diameter**, the Erlang application to which the commentary applies.

Warning:

The commentary is not yet complete. Comments currently stop at chapter 4.

```
Fajardo, et al.           Standards Track           [Page 6]
RFC 6733                 Diameter Base Protocol       October 2012

1. Introduction

Authentication, Authorization, and Accounting (AAA) protocols such as
TACACS [RFC1492] and RADIUS [RFC2865] were initially deployed to
provide dial-up PPP [RFC1661] and terminal server access. Over time,
AAA support was needed on many new access technologies, the scale and
complexity of AAA networks grew, and AAA was also used on new
applications (such as voice over IP). This led to new demands on AAA
protocols.
```

Note that diameter implements the Diameter protocol as defined in RFC 6733. It also supported the previous version of the protocol, as defined in RFC 3588, when there are differences. (Which will be noted below.) It does not support RADIUS.

```
Network access requirements for AAA protocols are summarized in
Aboba, et al. [RFC2989]. These include:
```

Failover

```
[RFC2865] does not define failover mechanisms and, as a result,
failover behavior differs between implementations. In order to
provide well-defined failover behavior, Diameter supports
application-layer acknowledgements and defines failover algorithms
and the associated state machine.
```

1.4 Standards Compliance

No comment.

Transmission-level security

RADIUS [RFC2865] defines an application-layer authentication and integrity scheme that is required only for use with response packets. While [RFC2869] defines an additional authentication and integrity mechanism, use is only required during Extensible Authentication Protocol (EAP) [RFC3748] sessions. While attribute hiding is supported, [RFC2865] does not provide support for per-packet confidentiality. In accounting, [RFC2866] assumes that replay protection is provided by the backend billing server rather than within the protocol itself.

While [RFC3162] defines the use of IPsec with RADIUS, support for IPsec is not required. In order to provide universal support for transmission-level security, and enable both intra- and inter-domain AAA deployments, Diameter provides support for TLS/TCP and DTLS/SCTP. Security is discussed in Section 13.

Whether or not IPsec is used is transparent to diameter.

The transport protocol used on a given peer connection is also transparent to diameter in that transport to diameter is simply a module that implements the transport protocol documented in *diameter_transport(3)*. A diameter user configures this module as the *diameter:transport_opt()* transport_module.

While a user can implement their own transport modules, diameter includes implementations for TCP and SCTP: *diameter_tcp(3)* based on *gen_tcp(3)* and *diameter_sctp(3)* based on *gen_sctp(3)*. The former supports TLS but the latter does not currently support DTLS.

Reliable transport

RADIUS runs over UDP, and does not define retransmission behavior; as a result, reliability varies between implementations. As described in [RFC2975], this is a major issue in accounting, where packet loss may translate directly into revenue loss. In order to

Fajardo, et al.

Standards Track

[Page 7]

RFC 6733

Diameter Base Protocol

October 2012

provide well-defined transport behavior, Diameter runs over reliable transport mechanisms (TCP, Stream Control Transmission Protocol (SCTP)) as defined in [RFC3539].

Agent support

RADIUS does not provide for explicit support for agents, including proxies, redirects, and relays. Since the expected behavior is not defined, it varies between implementations. Diameter defines agent behavior explicitly; this is described in Section 2.8.

No comment.

Server-initiated messages

While server-initiated messages are defined in RADIUS [RFC5176], support is optional. This makes it difficult to implement features such as unsolicited disconnect or re-authentication/re-authorization on demand across a heterogeneous deployment. To address this issue, support for server-initiated messages is mandatory in Diameter.

A diameter user can both send and receive messages.

Transition support

While Diameter does not share a common protocol data unit (PDU) with RADIUS, considerable effort has been expended in enabling backward compatibility with RADIUS so that the two protocols may be deployed in the same network. Initially, it is expected that Diameter will be deployed within new network devices, as well as within gateways enabling communication between legacy RADIUS devices and Diameter agents. This capability enables Diameter support to be added to legacy networks, by addition of a gateway or server speaking both RADIUS and Diameter.

RADIUS Attributes can be redefined as Diameter AVP's using diameter's *diameter_dict(4)* interface but diameter provides no such definitions.

In addition to addressing the above requirements, Diameter also provides support for the following:

Capability negotiation

RADIUS does not support error messages, capability negotiation, or a mandatory/non-mandatory flag for attributes. Since RADIUS clients and servers are not aware of each other's capabilities, they may not be able to successfully negotiate a mutually acceptable service or, in some cases, even be aware of what service has been implemented. Diameter includes support for error handling (Section 7), capability negotiation (Section 5.3), and mandatory/non-mandatory Attribute-Value Pairs (AVPs) (Section 4.1).

Peer discovery and configuration

RADIUS implementations typically require that the name or address

1.4 Standards Compliance

of servers or clients be manually configured, along with the corresponding shared secrets. This results in a large administrative burden and creates the temptation to reuse the RADIUS shared secret, which can result in major security vulnerabilities if the Request Authenticator is not globally and temporally unique as required in [RFC2865]. Through DNS, Diameter enables dynamic discovery of peers (see Section 5.2). Derivation of dynamic session keys is enabled via transmission-level security.

Over time, the capabilities of Network Access Server (NAS) devices have increased substantially. As a result, while Diameter is a considerably more sophisticated protocol than RADIUS, it remains feasible to implement it within embedded devices.

No comment.

1.1. Diameter Protocol

The Diameter base protocol provides the following facilities:

- o Ability to exchange messages and deliver AVPs

There are two interfaces directly involved in message exchange when using diameter: the function *diameter:call/4* for sending outgoing requests, and the application callback interface, documented in *diameter_app(3)* for receiving incoming request and answers.

- o Capabilities negotiation

Capabilities negotiation is the responsibility of diameter: a user configures a diameter service and/or transport with *capabilities* to provide AVP values for CER and CEA messages but it is diameter itself that sends these messages. A user receives notification of a successful capabilities exchange by way of *peer_up/3* callbacks.

- o Error notification

A user can subscribe to *events*, using *diameter:subscribe/1*, in order to receive notification of various failures. Errors in Diameter messaging are communicated via the application callbacks *handle_request/3*, *handle_answer/4* and *handle_error/4*.

- o Extensibility, required in [RFC2989], through addition of new applications, commands, and AVPs

Support for applications, commands and AVP's is extensible using diameter's dictionary interface, as documented in *diameter_dict(4)*. Dictionaries are compiled to Erlang encode/decode modules using *diameterc(1)* or *diameter_make(3)*.

- o Deprecated the use of the Inband-Security AVP for negotiating Transport Layer Security (TLS) [RFC5246]. It has been generally considered that bootstrapping of TLS via Inband-Security AVP creates certain security risks because it does not completely protect the information carried in the CER/CEA (Capabilities-Exchange-Request/Capabilities-Exchange-Answer). This version of Diameter adopts the common approach of defining a well-known secured port that peers should use when communicating via TLS/TCP and DTLS/SCTP. This new approach augments the existing in-band security negotiation, but it does not completely replace it. The old method is kept for backward compatibility reasons.

diameter_tcp(3) supports both methods of negotiating TLS: bootstrapping via Inband-Security and directly following connection establishment.

- o Deprecated the exchange of CER/CEA messages in the open state. This feature was implied in the peer state machine table of RFC 3588, but it was not clearly defined anywhere else in that document. As work on this document progressed, it became clear that the multiplicity of meaning and use of Application-Id AVPs in the CER/CEA messages (and the messages themselves) is seen as an abuse of the Diameter extensibility rules and thus required simplification. Capabilities exchange in the open state has been re-introduced in a separate specification [RFC6737], which clearly defines new commands for this feature.

Capabilities exchange in the open state is not supported: an incoming CER in the open state will cause diameter to ask the relevant transport process to terminate, which implies the loss of the peer connection in the case of *diameter_tcp(3)* and *diameter_sctp(3)*.

Capabilities update, as defined by RFC 6737, is not yet supported. Support will require diameter to handle CUR/CUA in the same way that it handles CER/CEA.

Fajardo, et al. Standards Track [Page 11]
RFC 6733 Diameter Base Protocol October 2012

- o Simplified security requirements. The use of a secured transport for exchanging Diameter messages remains mandatory. However, TLS/TCP and DTLS/SCTP have become the primary methods of securing Diameter with IPsec as a secondary alternative. See Section 13 for details. The support for the End-to-End security framework (E2E-Sequence AVP and 'P'-bit in the AVP header) has also been deprecated.

The End-to-End security framework is not supported since its use is largely unspecified: diameter will set the P-bit in outgoing AVP's as directed by the relevant dictionary and/or *prepare_request/3* or *handle_request/3* callbacks, but whether or not the P-bit is set on incoming AVP's has no consequence.

1.4 Standards Compliance

As noted above, DTLS is not currently supported and whether or not IPsec is used is transparent to diameter.

- o Changed Diameter extensibility. This includes fixes to the Diameter extensibility description (Section 1.3 and others) to better aid Diameter application designers; in addition, the new specification relaxes the policy with respect to the allocation of Command Codes for vendor-specific uses.
- o Clarified Application Id usage. Clarify the proper use of Application Id information, which can be found in multiple places within a Diameter message. This includes correlating Application Ids found in the message headers and AVPs. These changes also clearly specify the proper Application Id value to use for specific base protocol messages (ASR/ASA, STR/STA) as well as clarify the content and use of Vendor-Specific-Application-Id.
- o Clarified routing fixes. This document more clearly specifies what information (AVPs and Application Ids) can be used for making general routing decisions. A rule for the prioritization of redirect routing criteria when multiple route entries are found via redirects has also been added (see Section 6.13).
- o Simplified Diameter peer discovery. The Diameter discovery process now supports only widely used discovery schemes; the rest have been deprecated (see Section 5.2 for details).

Peer discover is not currently supported: peers to which a node should connect must be configured. Connection requests are accepted from arbitrary peers but a *diameter:transport_opt()* capabilities_cb can be used to reject a peer based on an incoming CER or CEA.

There are many other miscellaneous fixes that have been introduced in this document that may not be considered significant, but they have value nonetheless. Examples are removal of obsolete types, fixes to the state machine, clarification of the election process, message validation, fixes to Failed-AVP and Result-Code AVP values, etc. All of the errata filed against RFC 3588 prior to the publication of this document have been addressed. A comprehensive list of changes is not shown here for practical reasons.

1.2. Terminology

AAA

Authentication, Authorization, and Accounting.

Fajardo, et al.

Standards Track

[Page 12]

RFC 6733

Diameter Base Protocol

October 2012

ABNF

Augmented Backus-Naur Form [RFC5234]. A metalanguage with its own formal syntax and rules. It is based on the Backus-Naur Form and

is used to define message exchanges in a bi-directional communications protocol.

Accounting

The act of collecting information on resource usage for the purpose of capacity planning, auditing, billing, or cost allocation.

Accounting Record

An accounting record represents a summary of the resource consumption of a user over the entire session. Accounting servers creating the accounting record may do so by processing interim accounting events or accounting events from several devices serving the same user.

Authentication

The act of verifying the identity of an entity (subject).

Authorization

The act of determining whether a requesting entity (subject) will be allowed access to a resource (object).

Attribute-Value Pair (AVP)

The Diameter protocol consists of a header followed by one or more Attribute-Value-Pairs (AVPs). An AVP includes a header and is used to encapsulate protocol-specific data (e.g., routing information) as well as authentication, authorization, or accounting information.

No comment.

Command Code Format (CCF)

A modified form of ABNF used to define Diameter commands (see Section 3.2).

The @messages section of the *diameter_dict(4)* format has the CCF as content.

Diameter Agent

A Diameter Agent is a Diameter node that provides relay, proxy, redirect, or translation services.

Fajardo, et al.

Standards Track

[Page 13]

RFC 6733

Diameter Base Protocol

October 2012

Diameter Client

1.4 Standards Compliance

A Diameter client is a Diameter node that supports Diameter client applications as well as the base protocol. Diameter clients are often implemented in devices situated at the edge of a network and provide access control services for that network. Typical examples of Diameter clients include the Network Access Server (NAS) and the Mobile IP Foreign Agent (FA).

Diameter Node

A Diameter node is a host process that implements the Diameter protocol and acts as either a client, an agent, or a server.

Diameter Peer

Two Diameter nodes sharing a direct TCP or SCTP transport connection are called Diameter peers.

Diameter Server

A Diameter server is a Diameter node that handles authentication, authorization, and accounting requests for a particular realm. By its very nature, a Diameter server must support Diameter server applications in addition to the base protocol.

A Diameter Node is implemented by configuring a service using *diameter:start_service/2* and one or more transports using *diameter:add_transport/2*. The service typically represents a Diameter Node but since capabilities can be configured on individual transports it's more accurate to say that the node is a collection of transports advertising the same Origin-Host.

The role of a node (agent, client or server) is not something that's configured explicitly. Transports are either connecting or listening, depending on whether diameter should establish a peer connection and send CER or accept connections and receive CER, but the role a node implements depends largely on dictionary configuration and *diameter_app(3)* callback implementation.

Downstream

Downstream is used to identify the direction of a particular Diameter message from the home server towards the Diameter client.

Home Realm

A Home Realm is the administrative domain with which the user maintains an account relationship.

Home Server

A Diameter server that serves the Home Realm.

Interim Accounting

An interim accounting message provides a snapshot of usage during a user's session. Typically, it is implemented in order to provide for partial accounting of a user's session in case a device reboot or other network problem prevents the delivery of a session summary message or session record.

RFC 6733

Diameter Base Protocol

October 2012

Local Realm

A local realm is the administrative domain providing services to a user. An administrative domain may act as a local realm for certain users while being a home realm for others.

Multi-session

A multi-session represents a logical linking of several sessions. Multi-sessions are tracked by using the Acct-Multi-Session-Id. An example of a multi-session would be a Multi-link PPP bundle. Each leg of the bundle would be a session while the entire bundle would be a multi-session.

Network Access Identifier

The Network Access Identifier, or NAI [RFC4282], is used in the Diameter protocol to extract a user's identity and realm. The identity is used to identify the user during authentication and/or authorization while the realm is used for message routing purposes.

Proxy Agent or Proxy

In addition to forwarding requests and responses, proxies make policy decisions relating to resource usage and provisioning. Typically, this is accomplished by tracking the state of NAS devices. While proxies usually do not respond to client requests prior to receiving a response from the server, they may originate Reject messages in cases where policies are violated. As a result, proxies need to understand the semantics of the messages passing through them, and they may not support all Diameter applications.

Realm

The string in the NAI that immediately follows the '@' character. NAI realm names are required to be unique and are piggybacked on the administration of the DNS namespace. Diameter makes use of the realm, also loosely referred to as domain, to determine whether messages can be satisfied locally or whether they must be routed or redirected. In RADIUS, realm names are not necessarily piggybacked on the DNS namespace but may be independent of it.

Fajardo, et al.

Standards Track

[Page 15]

RFC 6733

Diameter Base Protocol

October 2012

Real-Time Accounting

Real-time accounting involves the processing of information on resource usage within a defined time window. Typically, time constraints are imposed in order to limit financial risk. The

1.4 Standards Compliance

Diameter Credit-Control Application [RFC4006] is an example of an application that defines real-time accounting functionality.

Relay Agent or Relay

Relays forward requests and responses based on routing-related AVPs and routing table entries. Since relays do not make policy decisions, they do not examine or alter non-routing AVPs. As a result, relays never originate messages, do not need to understand the semantics of messages or non-routing AVPs, and are capable of handling any Diameter application or message type. Since relays make decisions based on information in routing AVPs and realm forwarding tables, they do not keep state on NAS resource usage or sessions in progress.

Redirect Agent

Rather than forwarding requests and responses between clients and servers, redirect agents refer clients to servers and allow them to communicate directly. Since redirect agents do not sit in the forwarding path, they do not alter any AVPs transiting between client and server. Redirect agents do not originate messages and are capable of handling any message type, although they may be configured only to redirect messages of certain types, while acting as relay or proxy agents for other types. As with relay agents, redirect agents do not keep state with respect to sessions or NAS resources.

No comment.

Session

A session is a related progression of events devoted to a particular activity. Diameter application documents provide guidelines as to when a session begins and ends. All Diameter packets with the same Session-Id are considered to be part of the same session.

Sessions are not something that diameter is aware of. The function *diameter::session_id/1* can be used to construct appropriate values for Session-Id AVP's but logic connecting events in the same session is the responsibility of the diameter user.

Stateful Agent

A stateful agent is one that maintains session state information, by keeping track of all authorized active sessions. Each authorized session is bound to a particular service, and its state is considered active either until it is notified otherwise or until expiration.

Sub-session

A sub-session represents a distinct service (e.g., QoS or data characteristics) provided to a given session. These services may happen concurrently (e.g., simultaneous voice and data transfer during the same session) or serially. These changes in sessions are tracked with the Accounting-Sub-Session-Id.

Transaction State

The Diameter protocol requires that agents maintain transaction state, which is used for failover purposes. Transaction state implies that upon forwarding a request, the Hop-by-Hop Identifier is saved; the field is replaced with a locally unique identifier, which is restored to its original value when the corresponding answer is received. The request's state is released upon receipt of the answer. A stateless agent is one that only maintains transaction state.

Translation Agent

A translation agent (TLA in Figure 4) is a stateful Diameter node that performs protocol translation between Diameter and another AAA protocol, such as RADIUS.

Upstream

Upstream is used to identify the direction of a particular Diameter message from the Diameter client towards the home server.

User

The entity or device requesting or using some resource, in support of which a Diameter client has generated a request.

No comment.

1.3. Approach to Extensibility

The Diameter protocol is designed to be extensible, using several mechanisms, including:

- o Defining new AVP values
- o Creating new AVPs
- o Creating new commands
- o Creating new applications

Fajardo, et al.

Standards Track

[Page 17]

RFC 6733

Diameter Base Protocol

October 2012

From the point of view of extensibility, Diameter authentication, authorization, and accounting applications are treated in the same way.

1.4 Standards Compliance

Extensibility in diameter is by way of the dictionary interface documented in *diameter_dict(4)*: a diameter user creates applications, commands and AVP's by implementing a new dictionary, compiling the dictionary to a codec module using *diameterc(1)* or *diameter_make(3)*, and configuring the resulting dictionary module on a service. The dictionary modules provided with diameter are all implemented in this manner.

Note: Protocol designers should try to reuse existing functionality, namely AVP values, AVPs, commands, and Diameter applications. Reuse simplifies standardization and implementation. To avoid potential interoperability issues, it is important to ensure that the semantics of the reused features are well understood. Given that Diameter can also carry RADIUS attributes as Diameter AVPs, such reuse considerations also apply to existing RADIUS attributes that may be useful in a Diameter application.

Reuse in dictionary files is achieved by way of the `@inherits` section. AVP's are inherited, commands are not.

1.3.1. Defining New AVP Values

In order to allocate a new AVP value for AVPs defined in the Diameter base protocol, the IETF needs to approve a new RFC that describes the AVP value. IANA considerations for these AVP values are discussed in Section 11.3.

The allocation of AVP values for other AVPs is guided by the IANA considerations of the document that defines those AVPs. Typically, allocation of new values for an AVP defined in an RFC would require IETF Review [RFC5226], whereas values for vendor-specific AVPs can be allocated by the vendor.

1.3.2. Creating New AVPs

A new AVP being defined MUST use one of the data types listed in Sections 4.2 or 4.3. If an appropriate derived data type is already defined, it SHOULD be used instead of a base data type to encourage reusability and good design practice.

In the event that a logical grouping of AVPs is necessary, and multiple "groups" are possible in a given command, it is recommended that a Grouped AVP be used (see Section 4.4).

The creation of new AVPs can happen in various ways. The recommended approach is to define a new general-purpose AVP in a Standards Track RFC approved by the IETF. However, as described in Section 11.1.1, there are other mechanisms.

Creating new AVP's is an issue for the dictionary designer, not diameter.

1.3.3. Creating New Commands

A new Command Code MUST be allocated when required AVPs (those indicated as {AVP} in the CCF definition) are added to, deleted from, or redefined in (for example, by changing a required AVP into an optional one) an existing command.

message to another Diameter peer. The set of AVPs included in the message is determined by a particular Diameter application. One AVP that is included to reference a user's session is the Session-Id.

The initial request for authentication and/or authorization of a user would include the Session-Id AVP. The Session-Id is then used in all subsequent messages to identify the user's session (see Section 8 for

Fajardo, et al. Standards Track [Page 20]

RFC 6733 Diameter Base Protocol October 2012

more information). The communicating party may accept the request or reject it by returning an answer message with the Result-Code AVP set to indicate that an error occurred. The specific behavior of the Diameter server or client receiving a request depends on the Diameter application employed.

Session state (associated with a Session-Id) MUST be freed upon receipt of the Session-Termination-Request, Session-Termination-Answer, expiration of authorized service time in the Session-Timeout AVP, and according to rules established in a particular Diameter application.

Like Session-Id, session state is maintained by the diameter user: diameter has no session state of its own and does not interpret STR/STA in any way.

The base Diameter protocol may be used by itself for accounting applications. For authentication and authorization, it is always extended for a particular application.

Diameter clients MUST support the base protocol, which includes accounting. In addition, they MUST fully support each Diameter application that is needed to implement the client's service, e.g., Network Access Server Requirements (NASREQ) [RFC2881] and/or Mobile IPv4. A Diameter client MUST be referred to as "Diameter X Client" where X is the application that it supports and not a "Diameter Client".

Diameter servers MUST support the base protocol, which includes accounting. In addition, they MUST fully support each Diameter application that is needed to implement the intended service, e.g., NASREQ and/or Mobile IPv4. A Diameter server MUST be referred to as "Diameter X Server" where X is the application that it supports, and not a "Diameter Server".

Diameter relays and redirect agents are transparent to the Diameter applications, but they MUST support the Diameter base protocol, which includes accounting, and all Diameter applications.

Diameter proxies MUST support the base protocol, which includes accounting. In addition, they MUST fully support each Diameter application that is needed to implement proxied services, e.g., NASREQ and/or Mobile IPv4. A Diameter proxy MUST be referred to as "Diameter X Proxy" where X is the application which it supports, and not a "Diameter Proxy".

1.4 Standards Compliance

No comment.

```
Fajardo, et al.           Standards Track           [Page 21]
RFC 6733                 Diameter Base Protocol       October 2012
```

2.1. Transport

The Diameter Transport profile is defined in [RFC3539].

The base Diameter protocol is run on port 3868 for both TCP [RFC0793] and SCTP [RFC4960]. For TLS [RFC5246] and Datagram Transport Layer Security (DTLS) [RFC6347], a Diameter node that initiates a connection prior to any message exchanges MUST run on port 5658. It is assumed that TLS is run on top of TCP when it is used, and DTLS is run on top of SCTP when it is used.

Which port a transport connects to or listens on is a matter of configuration. Both *diameter_tcp(3)* and *diameter_sctp(3)* will default to 3868 if no other value is specified.

If the Diameter peer does not support receiving TLS/TCP and DTLS/SCTP connections on port 5658 (i.e., the peer complies only with RFC 3588), then the initiator MAY revert to using TCP or SCTP on port 3868. Note that this scheme is kept only for the purpose of backward compatibility and that there are inherent security vulnerabilities when the initial CER/CEA messages are sent unprotected (see Section 5.6).

Diameter clients MUST support either TCP or SCTP; agents and servers SHOULD support both.

A Diameter node MAY initiate connections from a source port other than the one that it declares it accepts incoming connections on, and it MUST always be prepared to receive connections on port 3868 for TCP or SCTP and port 5658 for TLS/TCP and DTLS/SCTP connections. When DNS-based peer discovery (Section 5.2) is used, the port numbers received from SRV records take precedence over the default ports (3868 and 5658).

A given Diameter instance of the peer state machine MUST NOT use more than one transport connection to communicate with a given peer, unless multiple instances exist on the peer, in which case a separate connection per process is allowed.

The *diameter:service_opt()* *restrict_connection* controls to what extent a diameter service allows multiple connections to the same peer. (As identified by the value of Origin-Host received from it during capabilities exchange.)

When no transport connection exists with a peer, an attempt to connect SHOULD be made periodically. This behavior is handled via the Tc timer (see Section 12 for details), whose recommended value is 30 seconds. There are certain exceptions to this rule, such as when a peer has terminated the transport connection stating that it does not wish to communicate.

The frequency of reconnection attempts is configured with the *diameter:transport_opt()* `connect_timer` and `watchdog_timer`.

When connecting to a peer and either zero or more transports are specified, TLS SHOULD be tried first, followed by DTLS, then by TCP, and finally by SCTP. See Section 5.2 for more information on peer discovery.

The order in which different transports are attempted depends on the order of *diameter:transport_opt()* `transport_module` and `transport_config` tuples in transport configuration.

Fajardo, et al.	Standards Track	[Page 22]
RFC 6733	Diameter Base Protocol	October 2012

Diameter implementations SHOULD be able to interpret ICMP protocol port unreachable messages as explicit indications that the server is not reachable, subject to security policy on trusting such messages. Further guidance regarding the treatment of ICMP errors can be found in [RFC5927] and [RFC5461]. Diameter implementations SHOULD also be able to interpret a reset from the transport and timed-out connection attempts. If Diameter receives data from the lower layer that cannot be parsed or identified as a Diameter error made by the peer, the stream is compromised and cannot be recovered. The transport connection MUST be closed using a RESET call (send a TCP RST bit) or an SCTP ABORT message (graceful closure is compromised).

ICMP messages and other transport-level errors aren't directly visible to diameter but transport implementations like *diameter_tcp(3)* and *diameter_sctp(3)* propagate these as terminating transport processes.

2.1.1. SCTP Guidelines

Diameter messages SHOULD be mapped into SCTP streams in a way that avoids head-of-the-line (HOL) blocking. Among different ways of performing the mapping that fulfill this requirement it is RECOMMENDED that a Diameter node send every Diameter message (request or response) over stream zero with the unordered flag set. However, Diameter nodes MAY select and implement other design alternatives for avoiding HOL blocking such as using multiple streams with the unordered flag cleared (as originally instructed in RFC 3588). On the receiving side, a Diameter entity MUST be ready to receive

1.4 Standards Compliance

Diameter messages over any stream, and it is free to return responses over a different stream. This way, both sides manage the available streams in the sending direction, independently of the streams chosen by the other side to send a particular Diameter message. These messages can be out-of-order and belong to different Diameter sessions.

diameter_sctp(3) allows the sender to specify a stream number explicitly. The stream on which an incoming message is received is passed to *handle_request/3* and *handle_answer/4* callbacks as *transport_data* in a *#diameter_packet{}*.

Ordered or unordered delivery can be configured per transport.

Out-of-order delivery has special concerns during a connection establishment and termination. When a connection is established, the responder side sends a CEA message and moves to R-Open state as specified in Section 5.6. If an application message is sent shortly after the CEA and delivered out-of-order, the initiator side, still in Wait-I-CEA state, will discard the application message and close the connection. In order to avoid this race condition, the receiver side SHOULD NOT use out-of-order delivery methods until the first message has been received from the initiator, proving that it has moved to I-Open state. To trigger such a message, the receiver side could send a DWR immediately after sending a CEA. Upon reception of the corresponding DWA, the receiver side should start using out-of-order delivery methods to counter the HOL blocking.

diameter_sctp(3) does not currently allow the user to switch between ordered and unordered delivery, or to specify the manner of sending per message: one or the other must be configured, the defaults being ordered.

Another race condition may occur when DPR and DPA messages are used. Both DPR and DPA are small in size; thus, they may be delivered to the peer faster than application messages when an out-of-order delivery mechanism is used. Therefore, it is possible that a DPR/DPA

Fajardo, et al. Standards Track [Page 23]
RFC 6733 Diameter Base Protocol October 2012

exchange completes while application messages are still in transit, resulting in a loss of these messages. An implementation could mitigate this race condition, for example, using timers, and wait for a short period of time for pending application level messages to arrive before proceeding to disconnect the transport connection. Eventually, lost messages are handled by the retransmission mechanism described in Section 5.5.4.

A Diameter agent SHOULD use dedicated payload protocol identifiers (PPIDs) for clear text and encrypted SCTP DATA chunks instead of only using the unspecified payload protocol identifier (value 0). For this purpose, two PPID values are allocated: the PPID value 46 is for Diameter messages in clear text SCTP DATA chunks, and the PPID value 47 is for Diameter messages in protected DTLS/SCTP DATA chunks.

1.4 Standards Compliance

These applications are implemented in the dictionary modules `diameter_gen_base_rfc6733`, `diameter_gen_acct_rfc6733` and `diameter_relay` respectively. There are also RFC 3588 versions or the common and accounting dictionaries: `diameter_gen_base_rfc3588` and `diameter_base_accounting`. (The inconsistent naming is historical.) Dictionary modules are configured using the `diameter:application_opt()` dictionary.

Relay and redirect agents MUST advertise the Relay Application ID, while all other Diameter nodes MUST advertise locally supported applications. The receiver of a Capabilities Exchange message advertising relay service MUST assume that the sender supports all current and future applications.

Diameter relay and proxy agents are responsible for finding an upstream server that supports the application of a particular message. If none can be found, an error message is returned with the Result-Code AVP set to `DIAMETER_UNABLE_TO_DELIVER`.

No comment.

2.5. Connections vs. Sessions

This section attempts to provide the reader with an understanding of the difference between "connection" and "session", which are terms used extensively throughout this document.

A connection refers to a transport-level connection between two peers that is used to send and receive Diameter messages. A session is a logical concept at the application layer that exists between the Diameter client and the Diameter server; it is identified via the Session-Id AVP.

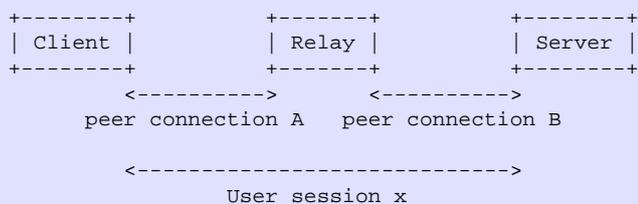


Figure 1: Diameter Connections and Sessions

In the example provided in Figure 1, peer connection A is established between the client and the relay. Peer connection B is established between the relay and the server. User session X spans from the client via the relay to the server. Each "user" of a service causes an auth request to be sent, with a unique session identifier. Once accepted by the server, both the client and the server are aware of the session.

It is important to note that there is no relationship between a

connection and a session, and that Diameter messages for multiple sessions are all multiplexed through a single connection. Also, note that Diameter messages pertaining to the session, both application-specific and those that are defined in this document such as ASR/ASA, RAR/RAA, and STR/STA, MUST carry the Application Id of the application. Diameter messages pertaining to peer connection establishment and maintenance such as CER/CEA, DWR/DWA, and DPR/DPA MUST carry an Application Id of zero (0).

As noted above, diameter is not involved in session management. This is the responsibility of the diameter user.

2.6. Peer Table

The Diameter peer table is used in message forwarding and is referenced by the routing table. A peer table entry contains the following fields:

Host Identity

Following the conventions described for the DiameterIdentity-derived AVP data format in Section 4.3.1, this field contains the contents of the Origin-Host (Section 6.3) AVP found in the CER or CEA message.

StatusT

This is the state of the peer entry, and it MUST match one of the values listed in Section 5.6.

Static or Dynamic

Specifies whether a peer entry was statically configured or dynamically discovered.

Expiration Time

Specifies the time at which dynamically discovered peer table entries are to be either refreshed or expired. If public key certificates are used for Diameter security (e.g., with TLS), this value MUST NOT be greater than the expiry times in the relevant certificates.

TLS/TCP and DTLS/SCTP Enabled

Specifies whether TLS/TCP and DTLS/SCTP is to be used when communicating with the peer.

Additional security information, when needed (e.g., keys, certificates).

The Peer Table is not directly accessible to the diameter user. Information about connected peers can be retrieved using *diameter:service_info/2*.

2.7. Routing Table

All Realm-Based routing lookups are performed against what is commonly known as the routing table (see Section 12). Each routing table entry contains the following fields:

Realm Name

This is the field that **MUST** be used as a primary key in the routing table lookups. Note that some implementations perform their lookups based on longest-match-from-the-right on the realm rather than requiring an exact match.

Application Identifier

An application is identified by an Application Id. A route entry can have a different destination based on the Application Id in the message header. This field **MUST** be used as a secondary key field in routing table lookups.

Local Action

The Local Action field is used to identify how a message should be treated. The following actions are supported:

1. LOCAL - Diameter messages that can be satisfied locally and do not need to be routed to another Diameter entity.
2. RELAY - All Diameter messages that fall within this category **MUST** be routed to a next-hop Diameter entity that is indicated by the identifier described below. Routing is done without modifying any non-routing AVPs. See Section 6.1.9 for relaying guidelines.
3. PROXY - All Diameter messages that fall within this category **MUST** be routed to a next Diameter entity that is indicated by the identifier described below. The local server **MAY** apply its local policies to the message by including new AVPs to the message prior to routing. See Section 6.1.9 for proxying guidelines.
4. REDIRECT - Diameter messages that fall within this category **MUST** have the identity of the home Diameter server(s) appended, and returned to the sender of the message. See Section 6.1.8 for redirection guidelines.

Server Identifier

The identity of one or more servers to which the message is to be routed. This identity **MUST** also be present in the Host Identity field of the peer table (Section 2.6). When the Local Action is set to RELAY or PROXY, this field contains the identity of the server(s) to which the message **MUST** be routed. When the Local

Action field is set to REDIRECT, this field contains the identity of one or more servers to which the message MUST be redirected.

Static or Dynamic

Specifies whether a route entry was statically configured or dynamically discovered.

Expiration Time

Specifies the time at which a dynamically discovered route table entry expires. If public key certificates are used for Diameter security (e.g., with TLS), this value MUST NOT be greater than the expiry time in the relevant certificates.

It is important to note that Diameter agents MUST support at least one of the LOCAL, RELAY, PROXY, or REDIRECT modes of operation. Agents do not need to support all modes of operation in order to conform with the protocol specification, but they MUST follow the protocol compliance guidelines in Section 2. Relay agents and proxies MUST NOT reorder AVPs.

The routing table MAY include a default entry that MUST be used for any requests not matching any of the other entries. The routing table MAY consist of only such an entry.

When a request is routed, the target server MUST have advertised the Application Id (see Section 2.4) for the given message or have advertised itself as a relay or proxy agent. Otherwise, an error is returned with the Result-Code AVP set to DIAMETER_UNABLE_TO_DELIVER.

Routing does not need specific support in diameter: a user can maintain their own routing table if desired and implement any desired routing in *diameter_app(3)* callbacks. However, it may be convenient to add more specific routing support to diameter in the future.

2.8. Role of Diameter Agents

In addition to clients and servers, the Diameter protocol introduces relay, proxy, redirect, and translation agents, each of which is defined in Section 1.2. Diameter agents are useful for several reasons:

As noted above, the role a node plays is largely a question of configuration and *diameter_app(3)* callback implementation.

- o They can distribute administration of systems to a configurable grouping, including the maintenance of security associations.

- o They can be used for concentration of requests from a number of

1.4 Standards Compliance

co-located or distributed NAS equipment sets to a set of like user groups.

- o They can do value-added processing to the requests or responses.
- o They can be used for load balancing.
- o A complex network will have multiple authentication sources, they can sort requests and forward towards the correct target.

The Diameter protocol requires that agents maintain transaction state, which is used for failover purposes. Transaction state implies that upon forwarding a request, its Hop-by-Hop Identifier is saved; the field is replaced with a locally unique identifier, which is restored to its original value when the corresponding answer is received. The request's state is released upon receipt of the answer. A stateless agent is one that only maintains transaction state.

The Proxy-Info AVP allows stateless agents to add local state to a Diameter request, with the guarantee that the same state will be present in the answer. However, the protocol's failover procedures require that agents maintain a copy of pending requests.

A stateful agent is one that maintains session state information by keeping track of all authorized active sessions. Each authorized session is bound to a particular service, and its state is considered active until either the agent is notified otherwise or the session expires. Each authorized session has an expiration, which is communicated by Diameter servers via the Session-Timeout AVP.

Maintaining session state may be useful in certain applications, such as:

- o Protocol translation (e.g., RADIUS <-> Diameter)
- o Limiting resources authorized to a particular user
- o Per-user or per-transaction auditing

A Diameter agent MAY act in a stateful manner for some requests and be stateless for others. A Diameter implementation MAY act as one type of agent for some requests and as another type of agent for others.

No comment.

Fajardo, et al. Standards Track [Page 29]

RFC 6733 Diameter Base Protocol October 2012

2.8.1. Relay Agents

Relay agents are Diameter agents that accept requests and route messages to other Diameter nodes based on information found in the

1.4 Standards Compliance

Proxies may, for example, be used in call control centers or access ISPs that provide outsourced connections; they can monitor the number and type of ports in use and make allocation and admission decisions according to their configuration.

Since enforcing policies requires an understanding of the service being provided, proxies **MUST** only advertise the Diameter applications they support.

No comment.

2.8.3. Redirect Agents

Redirect agents are useful in scenarios where the Diameter routing configuration needs to be centralized. An example is a redirect agent that provides services to all members of a consortium, but does not wish to be burdened with relaying all messages between realms. This scenario is advantageous since it does not require that the consortium provide routing updates to its members when changes are made to a member's infrastructure.

Since redirect agents do not relay messages, and only return an answer with the information necessary for Diameter agents to communicate directly, they do not modify messages. Since redirect agents do not receive answer messages, they cannot maintain session state.

The example provided in Figure 3 depicts a request issued from the access device, NAS, for the user bob@example.com. The message is forwarded by the NAS to its relay, DRL, which does not have a routing entry in its Diameter routing table for example.com. The DRL has a default route configured to DRD, which is a redirect agent that returns a redirect notification to DRL, as well as the HMS' contact information. Upon receipt of the redirect notification, the DRL establishes a transport connection with the HMS, if one doesn't already exist, and forwards the request to it.

Fajardo, et al.

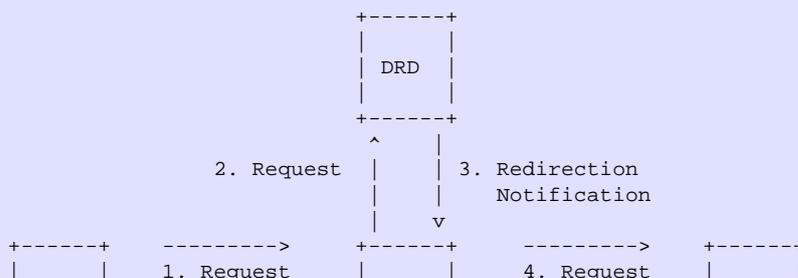
Standards Track

[Page 31]

RFC 6733

Diameter Base Protocol

October 2012



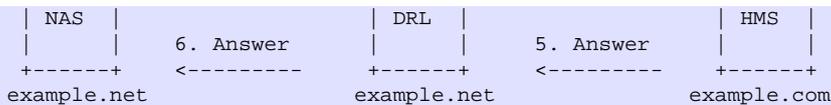


Figure 3: Redirecting a Diameter Message

Since redirect agents do not perform any application-level processing, they provide relaying services for all Diameter applications; therefore, they MUST advertise the Relay Application ID.

No comment.

2.8.4. Translation Agents

A translation agent is a device that provides translation between two protocols (e.g., RADIUS<->Diameter, TACACS+<->Diameter). Translation agents are likely to be used as aggregation servers to communicate with a Diameter infrastructure, while allowing for the embedded systems to be migrated at a slower pace.

Given that the Diameter protocol introduces the concept of long-lived authorized sessions, translation agents MUST be session stateful and MUST maintain transaction state.

Translation of messages can only occur if the agent recognizes the application of a particular request; therefore, translation agents MUST only advertise their locally supported applications.

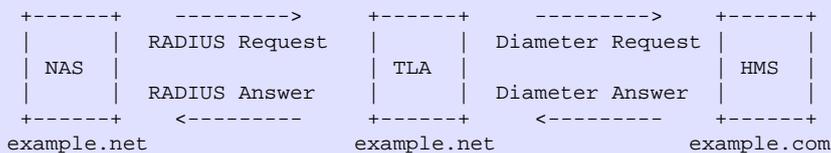


Figure 4: Translation of RADIUS to Diameter

No comment.

Fajardo, et al.

Standards Track

[Page 32]

RFC 6733

Diameter Base Protocol

October 2012

2.9. Diameter Path Authorization

As noted in Section 2.2, Diameter provides transmission-level security for each connection using TLS/TCP and DTLS/SCTP. Therefore, each connection can be authenticated and can be replay and integrity protected.

In addition to authenticating each connection, the entire session MUST also be authorized. Before initiating a connection, a Diameter

1.4 Standards Compliance

peer MUST check that its peers are authorized to act in their roles. For example, a Diameter peer may be authentic, but that does not mean that it is authorized to act as a Diameter server advertising a set of Diameter applications.

Prior to bringing up a connection, authorization checks are performed at each connection along the path. Diameter capabilities negotiation (CER/CEA) also MUST be carried out, in order to determine what Diameter applications are supported by each peer. Diameter sessions MUST be routed only through authorized nodes that have advertised support for the Diameter application required by the session.

As noted in Section 6.1.9, a relay or proxy agent MUST append a Route-Record AVP to all requests forwarded. The AVP contains the identity of the peer from which the request was received.

The home Diameter server, prior to authorizing a session, MUST check the Route-Record AVPs to make sure that the route traversed by the request is acceptable. For example, administrators within the home realm may not wish to honor requests that have been routed through an untrusted realm. By authorizing a request, the home Diameter server is implicitly indicating its willingness to engage in the business transaction as specified by any contractual relationship between the server and the previous hop. A DIAMETER_AUTHORIZATION_REJECTED error message (see Section 7.1.5) is sent if the route traversed by the request is unacceptable.

A home realm may also wish to check that each accounting request message corresponds to a Diameter response authorizing the session. Accounting requests without corresponding authorization responses SHOULD be subjected to further scrutiny, as should accounting requests indicating a difference between the requested and provided service.

Forwarding of an authorization response is considered evidence of a willingness to take on financial risk relative to the session. A local realm may wish to limit this exposure, for example, by establishing credit limits for intermediate realms and refusing to accept responses that would violate those limits. By issuing an

Fajardo, et al.

Standards Track

[Page 33]

RFC 6733

Diameter Base Protocol

October 2012

accounting request corresponding to the authorization response, the local realm implicitly indicates its agreement to provide the service indicated in the authorization response. If the service cannot be provided by the local realm, then a DIAMETER_UNABLE_TO_COMPLY error message MUST be sent within the accounting request; a Diameter client receiving an authorization response for a service that it cannot perform MUST NOT substitute an alternate service and then send accounting requests for the alternate service instead.

No comment.

3. Diameter Header

A summary of the Diameter header format is shown below. The fields

are transmitted in network byte order.

```

      0                1                2                3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  Version      |                Message Length      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Command Flags |                Command Code        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                Application-ID                      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                Hop-by-Hop Identifier              |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                End-to-End Identifier              |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  AVPs ...    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

The Diameter Header is represented by the `diameter_header` record defined in `diameter.hrl`. The `diameter_packet` record contains a header field whose value will be a decoded `#diameter_header{}` for incoming messages passed to `handle_request/3` and `handle_answer/4` callbacks. In the case of outgoing messages, diameter and the relevant dictionary populate the Diameter Header appropriately, although `prepare_request/3` and `handle_request/3` callbacks can modify header values. (Which can be useful in test.)

Version

This Version field MUST be set to 1 to indicate Diameter Version 1.

Message Length

The Message Length field is three octets and indicates the length of the Diameter message including the header fields and the padded AVPs. Thus, the Message Length field is always a multiple of 4.

Command Flags

The Command Flags field is eight bits. The following bits are assigned:

Fajardo, et al.

Standards Track

[Page 34]

RFC 6733

Diameter Base Protocol

October 2012

```

      0 1 2 3 4 5 6 7
+---+---+---+---+---+---+---+---+
|R P E T r r r r |
+---+---+---+---+---+---+---+---+

```

R(equest)

If set, the message is a request. If cleared, the message is an answer.

1.4 Standards Compliance

P(roxiable)

If set, the message MAY be proxied, relayed, or redirected. If cleared, the message MUST be locally processed.

E(rror)

If set, the message contains a protocol error, and the message will not conform to the CCF described for this command. Messages with the 'E' bit set are commonly referred to as error messages. This bit MUST NOT be set in request messages (see Section 7.2).

T(Potentially retransmitted message)

This flag is set after a link failover procedure, to aid the removal of duplicate requests. It is set when resending requests not yet acknowledged, as an indication of a possible duplicate due to a link failure. This bit MUST be cleared when sending a request for the first time; otherwise, the sender MUST set this flag. Diameter agents only need to be concerned about the number of requests they send based on a single received request; retransmissions by other entities need not be tracked. Diameter agents that receive a request with the T flag set, MUST keep the T flag set in the forwarded request. This flag MUST NOT be set if an error answer message (e.g., a protocol error) has been received for the earlier message. It can be set only in cases where no answer has been received from the server for a request, and the request has been sent again. This flag MUST NOT be set in answer messages.

r(eserved)

These flag bits are reserved for future use; they MUST be set to zero and ignored by the receiver.

Reserved bits are set to 0 in outgoing messages.

Fajardo, et al.

Standards Track

[Page 35]

RFC 6733

Diameter Base Protocol

October 2012

Command Code

The Command Code field is three octets and is used in order to communicate the command associated with the message. The 24-bit address space is managed by IANA (see Section 3.1). Command Code values 16,777,214 and 16,777,215 (hexadecimal values FFFFFE-FFFFFF) are reserved for experimental use (see Section 11.2).

Application-ID

Application-ID is four octets and is used to identify for which application the message is applicable. The application can be an authentication application, an accounting application, or a

1.4 Standards Compliance

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 the Diameter base protocol:

Command Name	Abbrev.	Code	Section Reference
Abort-Session-Request	ASR	274	8.5.1
Abort-Session-Answer	ASA	274	8.5.2
Accounting-Request	ACR	271	9.7.1
Accounting-Answer	ACA	271	9.7.2
Capabilities-Exchange-Request	CER	257	5.3.1
Capabilities-Exchange-Answer	CEA	257	5.3.2
Device-Watchdog-Request	DWR	280	5.5.1
Device-Watchdog-Answer	DWA	280	5.5.2
Disconnect-Peer-Request	DPR	282	5.4.1
Disconnect-Peer-Answer	DPA	282	5.4.2
Re-Auth-Request	RAR	258	8.3.1
Re-Auth-Answer	RAA	258	8.3.2
Session-Termination-Request	STR	275	8.4.1
Session-Termination-Answer	STA	275	8.4.2

These messages are all defined in diameter's implementation of the common dictionary in modules `diameter_gen_base_rfc6733` and `diameter_gen_base_rfc3588`. Corresponding record definitions are found in `diameter_gen_base_rfc6733.hrl` and `diameter_gen_base_rfc3588.hrl`.

Fajardo, et al. Standards Track [Page 37]
RFC 6733 Diameter Base Protocol October 2012

3.2. Command Code Format Specification

Every Command Code defined MUST include a corresponding Command Code Format (CCF) specification, which is used to define the AVPs that MUST or MAY be present when sending the message. The following ABNF specifies the CCF used in the definition:

The CCF is what is specified in the `@messages` section of the `diameter_dict(4)` format, except as noted below.

```
command-def = "<" command-name ">" "::=" diameter-message
```


1.4 Standards Compliance

```
qual          = [min] "*" [max]
               ; See ABNF conventions, RFC 5234, Section 4.
               ; The absence of any qualifier depends on
               ; whether it precedes a fixed, required, or
               ; optional rule.  If a fixed or required rule has
               ; no qualifier, then exactly one such AVP MUST
               ; be present.  If an optional rule has no
               ; qualifier, then 0 or 1 such AVP may be
               ; present.  If an optional rule has a qualifier,
               ; then the value of min MUST be 0 if present.

min           = 1*DIGIT
               ; The minimum number of times the element may
               ; be present.  If absent, the default value is 0
               ; for fixed and optional rules and 1 for
               ; required rules.  The value MUST be at least 1
               ; for required rules.

max           = 1*DIGIT
               ; The maximum number of times the element may
               ; be present.  If absent, the default value is
               ; infinity.  A value of 0 implies the AVP MUST
               ; NOT be present.

avp-spec      = diameter-name
               ; The avp-spec has to be an AVP Name, defined
               ; in the base or extended Diameter
               ; specifications.

avp-name      = avp-spec / "AVP"
               ; The string "AVP" stands for *any* arbitrary AVP
               ; Name, not otherwise listed in that Command Code
               ; definition.  The inclusion of this string
               ; is recommended for all CCFs to allow for
               ; extensibility.
```

Fajardo, et al. Standards Track [Page 39]
RFC 6733 Diameter Base Protocol October 2012

The following is a definition of a fictitious Command Code:

```
Example-Request ::= < Diameter Header: 9999999, REQ, PXY >
                 { User-Name }
                 1* { Origin-Host }
                 * [ AVP ]
```

No comment.

3.3. Diameter Command Naming Conventions

Diameter command names typically includes one or more English words followed by the verb "Request" or "Answer". Each English word is delimited by a hyphen. A three-letter acronym for both the request and answer is also normally provided.

An example is a message set used to terminate a session. The command

name is `Session-Terminate-Request` and `Session-Terminate-Answer`, while the acronyms are STR and STA, respectively.

Both the request and the answer for a given command share the same Command Code. The request is identified by the R(equest) bit in the Diameter header set to one (1), to ask that a particular action be performed, such as authorizing a user or terminating a session. Once the receiver has completed the request, it issues the corresponding answer, which includes a result code that communicates one of the following:

- o The request was successful
- o The request failed
- o An additional request has to be sent to provide information the peer requires prior to returning a successful or failed answer.
- o The receiver could not process the request, but provides information about a Diameter peer that is able to satisfy the request, known as redirect.

Additional information, encoded within AVPs, may also be included in answer messages.

The `diameter_dict(4)` format places no requirement on the naming of commands.

4. Diameter AVPs

Diameter AVPs carry specific authentication, accounting, authorization, and routing information as well as configuration details for the request and reply.

Fajardo, et al. Standards Track [Page 40]
RFC 6733 Diameter Base Protocol October 2012

Each AVP of type OctetString MUST be padded to align on a 32-bit boundary, while other AVP types align naturally. A number of zero-valued bytes are added to the end of the AVP Data field until a word boundary is reached. The length of the padding is not reflected in the AVP Length field.

4.1. AVP Header

The fields in the AVP header MUST be sent in network byte order. The format of the header is:

```

      0                1                2                3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
      |                                     AVP Code                                     |
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
      |V M P r r r r r|                                     AVP Length                                     |
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```


4.1.1. Optional Header Elements

The AVP header contains one optional field. This field is only present if the respective bit-flag is enabled.

Vendor-ID

The Vendor-ID field is present if the 'V' bit is set in the AVP Flags field. The optional four-octet Vendor-ID field contains the IANA-assigned "SMI Network Management Private Enterprise Codes" [ENTERPRISE] value, encoded in network byte order. Any vendors or standardization organizations that are also treated like vendors in the IANA-managed "SMI Network Management Private Enterprise Codes" space wishing to implement a vendor-specific Diameter AVP MUST use their own Vendor-ID along with their privately managed AVP address space, guaranteeing that they will not collide with any other vendor's vendor-specific AVP(s) or with future IETF AVPs.

Fajardo, et al.

Standards Track

[Page 42]

RFC 6733

Diameter Base Protocol

October 2012

A Vendor-ID value of zero (0) corresponds to the IETF-adopted AVP values, as managed by IANA. Since the absence of the Vendor-ID field implies that the AVP in question is not vendor specific, implementations MUST NOT use the value of zero (0) for the Vendor-ID field.

4.2. Basic AVP Data Formats

The Data field is zero or more octets and contains information specific to the Attribute. The format and length of the Data field is determined by the AVP Code and AVP Length fields. The format of the Data field MUST be one of the following base data types or a data type derived from the base data types. In the event that a new Basic AVP Data Format is needed, a new version of this RFC MUST be created.

OctetString

The data contains arbitrary data of variable length. Unless otherwise noted, the AVP Length field MUST be set to at least 8 (12 if the 'V' bit is enabled). AVP values of this type that are not a multiple of 4 octets in length are followed by the necessary padding so that the next AVP (if any) will start on a 32-bit boundary.

Integer32

32-bit signed value, in network byte order. The AVP Length field MUST be set to 12 (16 if the 'V' bit is enabled).

Integer64

64-bit signed value, in network byte order. The AVP Length field MUST be set to 16 (20 if the 'V' bit is enabled).

Unsigned32

32-bit unsigned value, in network byte order. The AVP Length field MUST be set to 12 (16 if the 'V' bit is enabled).

Unsigned64

64-bit unsigned value, in network byte order. The AVP Length field MUST be set to 16 (20 if the 'V' bit is enabled).

Fajardo, et al. Standards Track [Page 43]
RFC 6733 Diameter Base Protocol October 2012

Float32

This represents floating point values of single precision as described by [FLOATPOINT]. The 32-bit value is transmitted in network byte order. The AVP Length field MUST be set to 12 (16 if the 'V' bit is enabled).

Float64

This represents floating point values of double precision as described by [FLOATPOINT]. The 64-bit value is transmitted in network byte order. The AVP Length field MUST be set to 16 (20 if the 'V' bit is enabled).

Grouped

The Data field is specified as a sequence of AVPs. These AVPs are concatenated -- including their headers and padding -- in the order in which they are specified and the result encapsulated in the Data field. The AVP Length field is set to 8 (12 if the 'V' bit is enabled) plus the total length of all included AVPs, including their headers and padding. Thus, the AVP Length field of an AVP of type Grouped is always a multiple of 4.

4.3. Derived AVP Data Formats

In addition to using the Basic AVP Data Formats, applications may define data formats derived from the Basic AVP Data Formats. An application that defines new Derived AVP Data Formats MUST include them in a section titled "Derived AVP Data Formats", using the same format as the definitions below. Each new definition MUST be either defined or listed with a reference to the RFC that defines the format.

4.3.1. Common Derived AVP Data Formats

The following are commonly used Derived AVP Data Formats.

Address

The Address format is derived from the OctetString Basic AVP Format. It is a discriminated union representing, for example, a 32-bit (IPv4) [RFC0791] or 128-bit (IPv6) [RFC4291] address, most significant octet first. The first two octets of the Address AVP represent the AddressType, which contains an Address Family,

defined in [IANAADFAM]. The AddressType is used to discriminate the content and format of the remaining octets.

Fajardo, et al. Standards Track [Page 44]
RFC 6733 Diameter Base Protocol October 2012

Time

The Time format is derived from the OctetString Basic AVP Format. The string MUST contain four octets, in the same format as the first four bytes are in the NTP timestamp format. The NTP timestamp format is defined in Section 3 of [RFC5905].

This represents the number of seconds since 0h on 1 January 1900 with respect to the Coordinated Universal Time (UTC).

On 6h 28m 16s UTC, 7 February 2036, the time value will overflow. Simple Network Time Protocol (SNTP) [RFC5905] describes a procedure to extend the time to 2104. This procedure MUST be supported by all Diameter nodes.

UTF8String

The UTF8String format is derived from the OctetString Basic AVP Format. This is a human-readable string represented using the ISO/IEC IS 10646-1 character set, encoded as an OctetString using the UTF-8 transformation format [RFC3629].

Since additional code points are added by amendments to the 10646 standard from time to time, implementations MUST be prepared to encounter any code point from 0x00000001 to 0x7fffffff. Byte sequences that do not correspond to the valid encoding of a code point into UTF-8 charset or are outside this range are prohibited.

The use of control codes SHOULD be avoided. When it is necessary to represent a new line, the control code sequence CR LF SHOULD be used.

The use of leading or trailing white space SHOULD be avoided.

For code points not directly supported by user interface hardware or software, an alternative means of entry and display, such as hexadecimal, MAY be provided.

For information encoded in 7-bit US-ASCII, the UTF-8 charset is identical to the US-ASCII charset.

UTF-8 may require multiple bytes to represent a single character / code point; thus, the length of a UTF8String in octets may be different from the number of characters encoded.

Note that the AVP Length field of an UTF8String is measured in octets not characters.

Fajardo, et al. Standards Track [Page 45]
RFC 6733 Diameter Base Protocol October 2012

1.4 Standards Compliance

DiameterIdentity

The DiameterIdentity format is derived from the OctetString Basic AVP Format.

DiameterIdentity = FQDN/Realm

The DiameterIdentity value is used to uniquely identify either:

- * A Diameter node for purposes of duplicate connection and routing loop detection.
- * A Realm to determine whether messages can be satisfied locally or whether they must be routed or redirected.

When a DiameterIdentity value is used to identify a Diameter node, the contents of the string MUST be the Fully Qualified Domain Name (FQDN) of the Diameter node. If multiple Diameter nodes run on the same host, each Diameter node MUST be assigned a unique DiameterIdentity. If a Diameter node can be identified by several FQDNs, a single FQDN should be picked at startup and used as the only DiameterIdentity for that node, whatever the connection on which it is sent. In this document, note that DiameterIdentity is in ASCII form in order to be compatible with existing DNS infrastructure. See Appendix D for interactions between the Diameter protocol and Internationalized Domain Names (IDNs).

DiameterURI

The DiameterURI MUST follow the Uniform Resource Identifiers (RFC 3986) syntax [RFC3986] rules specified below:

"aaa://" FQDN [port] [transport] [protocol]

; No transport security

"aaas://" FQDN [port] [transport] [protocol]

; Transport security used

FQDN = < Fully Qualified Domain Name >

Fajardo, et al.

Standards Track

[Page 46]

RFC 6733

Diameter Base Protocol

October 2012

port = ":" 1*DIGIT

; One of the ports used to listen for
; incoming connections.
; If absent, the default Diameter port
; (3868) is assumed if no transport
; security is used and port 5658 when

```

; transport security (TLS/TCP and DTLS/SCTP)
; is used.

transport          = ";transport=" transport-protocol

; One of the transports used to listen
; for incoming connections.  If absent,
; the default protocol is assumed to be TCP.
; UDP MUST NOT be used when the aaa-protocol
; field is set to diameter.

transport-protocol = ( "tcp" / "sctp" / "udp" )

protocol          = ";protocol=" aaa-protocol

; If absent, the default AAA protocol
; is Diameter.

aaa-protocol      = ( "diameter" / "radius" / "tacacs+" )

```

The following are examples of valid Diameter host identities:

```

aaa://host.example.com;transport=tcp
aaa://host.example.com:6666;transport=tcp
aaa://host.example.com;protocol=diameter
aaa://host.example.com:6666;protocol=diameter
aaa://host.example.com:6666;transport=tcp;protocol=diameter
aaa://host.example.com:1813;transport=udp;protocol=radius

```

Enumerated

The Enumerated format is derived from the Integer32 Basic AVP Format. The definition contains a list of valid values and their interpretation and is described in the Diameter application introducing the AVP.

Fajardo, et al. Standards Track [Page 47]
RFC 6733 Diameter Base Protocol October 2012

IPFilterRule

The IPFilterRule format is derived from the OctetString Basic AVP Format and uses the ASCII charset. The rule syntax is a modified subset of ipfw(8) from FreeBSD. Packets may be filtered based on the following information that is associated with it:

```

Direction                              (in or out)
Source and destination IP address      (possibly masked)
Protocol
Source and destination port            (lists or ranges)
TCP flags
IP fragment flag
IP options
ICMP types

```

1.4 Standards Compliance

Rules for the appropriate direction are evaluated in order, with the first matched rule terminating the evaluation. Each packet is evaluated once. If no rule matches, the packet is dropped if the last rule evaluated was a permit, and passed if the last rule was a deny.

IPFilterRule filters MUST follow the format:

```
action dir proto from src to dst [options]

action      permit - Allow packets that match the rule.
            deny  - Drop packets that match the rule.

dir         "in" is from the terminal, "out" is to the
            terminal.

proto       An IP protocol specified by number. The "ip"
            keyword means any protocol will match.

src and dst <address/mask> [ports]

            The <address/mask> may be specified as:
            ipno      An IPv4 or IPv6 number in dotted-
                       quad or canonical IPv6 form. Only
                       this exact IP number will match the
                       rule.
```

Fajardo, et al.

Standards Track

[Page 48]

RFC 6733

Diameter Base Protocol

October 2012

```
ipno/bits  An IP number as above with a mask
            width of the form 192.0.2.10/24. In
            this case, all IP numbers from
            192.0.2.0 to 192.0.2.255 will match.
            The bit width MUST be valid for the
            IP version, and the IP number MUST
            NOT have bits set beyond the mask.
            For a match to occur, the same IP
            version must be present in the
            packet that was used in describing
            the IP address. To test for a
            particular IP version, the bits part
            can be set to zero. The keyword
            "any" is 0.0.0.0/0 or the IPv6
            equivalent. The keyword "assigned"
            is the address or set of addresses
            assigned to the terminal. For IPv4,
            a typical first rule is often "deny
            in ip! assigned".
```

The sense of the match can be inverted by preceding an address with the not modifier (!), causing all other addresses to be matched instead. This does not affect the selection of port numbers.

With the TCP, UDP, and SCTP protocols, optional ports may be specified as:

```
{port/port-port}[,ports[,...]]
```

The '-' notation specifies a range of ports (including boundaries).

Fragmented packets that have a non-zero offset (i.e., not the first fragment) will never match a rule that has one or more port specifications. See the frag option for details on matching fragmented packets.

options:

frag Match if the packet is a fragment and this is not the first fragment of the datagram. frag may not be used in conjunction with either tcpflags or TCP/UDP port specifications.

Fajardo, et al.

Standards Track

[Page 49]

RFC 6733

Diameter Base Protocol

October 2012

ipoptions spec

Match if the IP header contains the comma-separated list of options specified in spec. The supported IP options are:

ssrr (strict source route), lsrr (loose source route), rr (record packet route), and ts (timestamp). The absence of a particular option may be denoted with a '!'.

tcptoptions spec

Match if the TCP header contains the comma-separated list of options specified in spec. The supported TCP options are:

mss (maximum segment size), window (tcp window advertisement), sack (selective ack), ts (rfc1323 timestamp), and cc (rfc1644 t/tcp connection count). The absence of a particular option may be denoted with a '!'.

established

TCP packets only. Match packets that have the RST or ACK bits set.

setup

TCP packets only. Match packets that have the SYN bit set but no ACK bit.

tcpflags spec

TCP packets only. Match if the TCP header contains the comma-separated list of flags specified in spec. The supported TCP flags are:

fin, syn, rst, psh, ack, and urg. The absence of a particular flag may be denoted with a '!'. A rule that contains a tcpflags specification can never match a fragmented packet that has a non-zero offset. See the frag option for details on matching fragmented packets.

icmptypes types

ICMP packets only. Match if the ICMP type is in the list types. The list may be specified as any combination of ranges or individual types separated by commas. Both the numeric values and the symbolic values listed below can be used. The supported ICMP types are:

Fajardo, et al. Standards Track [Page 50]
RFC 6733 Diameter Base Protocol October 2012

echo reply (0), destination unreachable (3), source quench (4), redirect (5), echo request (8), router advertisement (9), router solicitation (10), time-to-live exceeded (11), IP header bad (12), timestamp request (13), timestamp reply (14), information request (15), information reply (16), address mask request (17), and address mask reply (18).

There is one kind of packet that the access device MUST always discard, that is an IP fragment with a fragment offset of one. This is a valid packet, but it only has one use, to try to circumvent firewalls.

An access device that is unable to interpret or apply a deny rule MUST terminate the session. An access device that is unable to interpret or apply a permit rule MAY apply a more restrictive rule. An access device MAY apply deny rules of its own before the supplied rules, for example to protect the access device owner's infrastructure.

4.4. Grouped AVP Values

The Diameter protocol allows AVP values of type 'Grouped'. This implies that the Data field is actually a sequence of AVPs. It is possible to include an AVP with a Grouped type within a Grouped type, that is, to nest them. AVPs within an AVP of type Grouped have the same padding requirements as non-Grouped AVPs, as defined in Section 4.4.

The AVP Code numbering space of all AVPs included in a Grouped AVP is the same as for non-Grouped AVPs. Receivers of a Grouped AVP that does not have the 'M' (mandatory) bit set and one or more of the encapsulated AVPs within the group has the 'M' (mandatory) bit set MAY simply be ignored if the Grouped AVP itself is unrecognized. The rule applies even if the encapsulated AVP with its 'M' (mandatory) bit set is further encapsulated within other sub-groups, i.e., other Grouped AVPs embedded within the Grouped AVP.

Every Grouped AVP definition MUST include a corresponding grammar, using ABNF [RFC5234] (with modifications), as defined below.

```
grouped-avp-def = "<" name ">" "::=" avp
```

```
name-fmt          = ALPHA *(ALPHA / DIGIT / "-")
```

Fajardo, et al. Standards Track [Page 51]

RFC 6733 Diameter Base Protocol October 2012

```
name              = name-fmt
                  ; The name has to be the name of an AVP,
                  ; defined in the base or extended Diameter
                  ; specifications.

avp               = header *fixed *required *optional

header            = "<" "AVP-Header:" avpcode [vendor] ">"

avpcode           = 1*DIGIT
                  ; The AVP Code assigned to the Grouped AVP.

vendor            = 1*DIGIT
                  ; The Vendor-ID assigned to the Grouped AVP.
                  ; If absent, the default value of zero is
                  ; used.
```

4.4.1. Example AVP with a Grouped Data Type

The Example-AVP (AVP Code 999999) is of type Grouped and is used to clarify how Grouped AVP values work. The Grouped Data field has the following CCF grammar:

```
Example-AVP ::= < AVP Header: 999999 >
              { Origin-Host }
              1*{ Session-Id }
              *[ AVP ]
```

An Example-AVP with Grouped Data follows.

The Origin-Host AVP (Section 6.3) is required. In this case:

```
Origin-Host = "example.com".
```

One or more Session-Ids must follow. Here there are two:

```
Session-Id =
  "grump.example.com:33041;23432;893;0AF3B81"
```

```
Session-Id =
  "grump.example.com:33054;23561;2358;0AF3B82"
```

Fajardo, et al. Standards Track [Page 52]

1.4 Standards Compliance

RFC 6733

Diameter Base Protocol

October 2012

optional AVPs included are

Recovery-Policy = <binary>

```
2163bc1d0ad82371f6bc09484133c3f09ad74a0dd5346d54195a7cf0b35
2cabc881839a4fcdcfbc1769e2677a4c1fb499284c5f70b48f58503a45c5
c2d6943f82d5930f2b7c1da640f476f0e9c9572a50db8ea6e51e1c2c7bd
f8bb43dc995144b8dbe297ac739493946803e1cee3e15d9b765008alb2a
cf4ac777c80041d72c01e691cf751dbf86e85f509f3988e5875dc905119
26841f00f0e29a6d1ddc1a842289d440268681e052b30fb638045f7779c
1d873c784f054f688f5001559ecff64865ef975f3e60d2fd7966b8c7f92
```

Futuristic-Acct-Record = <binary>

```
fe19da5802acd98b07a5b86cb4d5d03f0314ab9ef1ad0b67111ff3b90a0
57fe29620bf3585fd2dd9fcc38ce62f6cc208c6163c008f4258d1bc88b8
17694a74ccad3ec69269461b14b2e7a4c111fb239e33714da207983f58c
41d018d56fe938f3cbf089aac12a912a2f0d1923a9390e5f789cb2e5067
d3427475e49968f841
```

The data for the optional AVPs is represented in hexadecimal form since the format of these AVPs is not known at the time of definition of the Example-AVP group nor (likely) at the time when the example instance of this AVP is interpreted -- except by Diameter implementations that support the same set of AVPs. The encoding example illustrates how padding is used and how length fields are calculated. Also, note that AVPs may be present in the Grouped AVP value that the receiver cannot interpret (here, the Recover-Policy and Futuristic-Acct-Record AVPs). The length of the Example-AVP is the sum of all the length of the member AVPs, including their padding, plus the Example-AVP header size.

Fajardo, et al.

Standards Track

[Page 53]

RFC 6733

Diameter Base Protocol

October 2012

This AVP would be encoded as follows:

```
      0       1       2       3       4       5       6       7
+-----+-----+-----+-----+-----+-----+-----+
0 |           Example AVP Header (AVP Code = 999999), Length = 496           |
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
8 |      Origin-Host AVP Header (AVP Code = 264), Length = 19 |
+-----+-----+-----+-----+-----+-----+-----+-----+
16 | 'e' | 'x' | 'a' | 'm' | 'p' | 'l' | 'e' | '.' |
+-----+-----+-----+-----+-----+-----+-----+-----+
24 | 'c' | 'o' | 'm' | Padding |      Session-Id AVP Header |
+-----+-----+-----+-----+-----+-----+-----+-----+
32 | (AVP Code = 263), Length = 49 | 'g' | 'r' | 'u' | 'm' |
+-----+-----+-----+-----+-----+-----+-----+-----+
      . . .
+-----+-----+-----+-----+-----+-----+-----+-----+
72 | 'F' | '3' | 'B' | '8' | '1' | Padding | Padding | Padding |
+-----+-----+-----+-----+-----+-----+-----+-----+
80 |      Session-Id AVP Header (AVP Code = 263), Length = 50 |
+-----+-----+-----+-----+-----+-----+-----+-----+
88 | 'g' | 'r' | 'u' | 'm' | 'p' | '.' | 'e' | 'x' |
+-----+-----+-----+-----+-----+-----+-----+-----+
      . . .
+-----+-----+-----+-----+-----+-----+-----+-----+
120 | '5' | '8' | ';' | '0' | 'A' | 'F' | '3' | 'B' |
+-----+-----+-----+-----+-----+-----+-----+-----+
128 | '8' | '2' | Padding | Padding | Recovery-Policy Header (AVP |
+-----+-----+-----+-----+-----+-----+-----+-----+
136 | Code = 8341), Length = 223 | 0x21 | 0x63 | 0xbc | 0x1d |
+-----+-----+-----+-----+-----+-----+-----+-----+
144 | 0x0a | 0xd8 | 0x23 | 0x71 | 0xf6 | 0xbc | 0x09 | 0x48 |
+-----+-----+-----+-----+-----+-----+-----+-----+
      . . .
+-----+-----+-----+-----+-----+-----+-----+-----+
352 | 0x8c | 0x7f | 0x92 | Padding | Futuristic-Acct-Record Header |
+-----+-----+-----+-----+-----+-----+-----+-----+
328 | (AVP Code = 15930), Length = 137 | 0xfe | 0x19 | 0xda | 0x58 |
+-----+-----+-----+-----+-----+-----+-----+-----+
336 | 0x02 | 0xac | 0xd9 | 0x8b | 0x07 | 0xa5 | 0xb8 | 0xc6 |
+-----+-----+-----+-----+-----+-----+-----+-----+
      . . .
+-----+-----+-----+-----+-----+-----+-----+-----+
488 | 0xe4 | 0x99 | 0x68 | 0xf8 | 0x41 | Padding | Padding | Padding |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Fajardo, et al.

Standards Track

[Page 54]

RFC 6733

Diameter Base Protocol

October 2012

4.5. Diameter Base Protocol AVPs

The following table describes the Diameter AVPs defined in the base protocol, their AVP Code values, types, and possible flag values.

Due to space constraints, the short form DiamIdent is used to represent DiameterIdentity.

Attribute Name	AVP Code	Section Defined	Data Type	AVP Flag rules	
				MUST	NOT
Acct-Interim-Interval	85	9.8.2	Unsigned32	M	V
Accounting-Realtime-Required	483	9.8.7	Enumerated	M	V
Acct-Multi-Session-Id	50	9.8.5	UTF8String	M	V
Accounting-Record-Number	485	9.8.3	Unsigned32	M	V
Accounting-Record-Type	480	9.8.1	Enumerated	M	V
Acct-Session-Id	44	9.8.4	OctetString	M	V
Accounting-Sub-Session-Id	287	9.8.6	Unsigned64	M	V
Acct-Application-Id	259	6.9	Unsigned32	M	V

Auth-Application-Id	258	6.8	Unsigned32	M	V
Auth-Request-Type	274	8.7	Enumerated	M	V
Authorization-Lifetime	291	8.9	Unsigned32	M	V
Auth-Grace-Period	276	8.10	Unsigned32	M	V
Auth-Session-State	277	8.11	Enumerated	M	V
Re-Auth-Request-Type	285	8.12	Enumerated	M	V
Class	25	8.20	OctetString	M	V
Destination-Host	293	6.5	DiamIdent	M	V
Destination-Realm	283	6.6	DiamIdent	M	V
Disconnect-Cause	273	5.4.3	Enumerated	M	V
Error-Message	281	7.3	UTF8String		V,M
Error-Reporting-Host	294	7.4	DiamIdent		V,M
Event-Timestamp	55	8.21	Time	M	V
Experimental-Result	297	7.6	Grouped	M	V

Fajardo, et al.

Standards Track

[Page 56]

RFC 6733

Diameter Base Protocol

October 2012

Attribute Name	AVP Code	Section Defined	Data Type	AVP Flag rules	
				MUST	NOT
Experimental-Result-Code	298	7.7	Unsigned32	M	V
Failed-AVP	279	7.5	Grouped	M	V
Firmware-Revision	267	5.3.4	Unsigned32		V,M
Host-IP-Address	257	5.3.5	Address	M	V
Inband-Security-Id	299	6.10	Unsigned32	M	V
Multi-Round-Time-Out	272	8.19	Unsigned32	M	V
Origin-Host	264	6.3	DiamIdent	M	V
Origin-Realm	296	6.4	DiamIdent	M	V
Origin-State-Id	278	8.16	Unsigned32	M	V
Product-Name	269	5.3.7	UTF8String		V,M
Proxy-Host	280	6.7.3	DiamIdent	M	V
Proxy-Info	284	6.7.2	Grouped	M	V
Proxy-State	33	6.7.4	OctetString	M	V
Redirect-Host	292	6.12	DiamURI	M	V
Redirect-Host-Usage	261	6.13	Enumerated	M	V
Redirect-Max-Cache-Time	262	6.14	Unsigned32	M	V
Result-Code	268	7.1	Unsigned32	M	V
Route-Record	282	6.7.1	DiamIdent	M	V
Session-Id	263	8.8	UTF8String	M	V

1.4 Standards Compliance

Session-Timeout	27	8.13	Unsigned32	M	V
Session-Binding	270	8.17	Unsigned32	M	V
Session-Server-Failover	271	8.18	Enumerated	M	V
Supported-Vendor-Id	265	5.3.6	Unsigned32	M	V
Termination-Cause	295	8.15	Enumerated	M	V
User-Name	1	8.14	UTF8String	M	V
Vendor-Id	266	5.3.3	Unsigned32	M	V
Vendor-Specific-Application-Id	260	6.11	Grouped	M	V

Fajardo, et al.

Standards Track

[Page 57]

RFC 6733

Diameter Base Protocol

October 2012

5. Diameter Peers

This section describes how Diameter nodes establish connections and communicate with peers.

5.1. Peer Connections

Connections between diameter peers are established using their valid DiameterIdentity. A Diameter node initiating a connection to a peer MUST know the peer's DiameterIdentity. Methods for discovering a Diameter peer can be found in Section 5.2.

Although a Diameter node may have many possible peers with which it is able to communicate, it may not be economical to have an established connection to all of them. At a minimum, a Diameter node SHOULD have an established connection with two peers per realm, known as the primary and secondary peers. Of course, a node MAY have additional connections, if it is deemed necessary. Typically, all messages for a realm are sent to the primary peer but, in the event that failover procedures are invoked, any pending requests are sent to the secondary peer. However, implementations are free to load balance requests between a set of peers.

Note that a given peer MAY act as a primary for a given realm while acting as a secondary for another realm.

When a peer is deemed suspect, which could occur for various reasons, including not receiving a DWA within an allotted time frame, no new requests should be forwarded to the peer, but failover procedures are invoked. When an active peer is moved to this mode, additional connections SHOULD be established to ensure that the necessary number of active connections exists.

There are two ways that a peer is removed from the suspect peer list:

1. The peer is no longer reachable, causing the transport connection to be shut down. The peer is moved to the closed state.
2. Three watchdog messages are exchanged with accepted round-trip times, and the connection to the peer is considered stabilized.

Fajardo, et al. Standards Track [Page 59]

RFC 6733 Diameter Base Protocol October 2012

RR's) for the target hostname specified in the SRV records following the rules given in [RFC2782]. If no SRV records are found, the requester gives up.

If the server is using a site certificate, the domain name in the NAPTR query and the domain name in the replacement field MUST both be valid based on the site certificate handed out by the server in the TLS/TCP and DTLS/SCTP or Internet Key Exchange Protocol (IKE) exchange. Similarly, the domain name in the SRV query and the domain name in the target in the SRV record MUST both be valid based on the same site certificate. Otherwise, an attacker could modify the DNS records to contain replacement values in a different domain, and the client could not validate whether this was the desired behavior or the result of an attack.

Also, the Diameter peer MUST check to make sure that the discovered peers are authorized to act in its role. Authentication via IKE or TLS/TCP and DTLS/SCTP, or validation of DNS RRs via DNSSEC is not sufficient to conclude this. For example, a web server may have obtained a valid TLS/TCP and DTLS/SCTP certificate, and secured RRs may be included in the DNS, but this does not imply that it is authorized to act as a Diameter server.

Authorization can be achieved, for example, by the configuration of a Diameter server Certification Authority (CA). The server CA issues a certificate to the Diameter server, which includes an Object Identifier (OID) to indicate the subject is a Diameter server in the Extended Key Usage extension [RFC5280]. This certificate is then used during TLS/TCP, DTLS/SCTP, or IKE security negotiation. However, note that, at the time of writing, no Diameter server Certification Authorities exist.

A dynamically discovered peer causes an entry in the peer table (see Section 2.6) to be created. Note that entries created via DNS MUST expire (or be refreshed) within the DNS Time to Live (TTL). If a peer is discovered outside of the local realm, a routing table entry (see Section 2.7) for the peer's realm is created. The routing table entry's expiration MUST match the peer's expiration value.

5.3. Capabilities Exchange

When two Diameter peers establish a transport connection, they MUST exchange the Capabilities Exchange messages, as specified in the peer state machine (see Section 5.6). This message allows the discovery of a peer's identity and its capabilities (protocol version number, the identifiers of supported Diameter applications, security mechanisms, etc.).

Fajardo, et al. Standards Track [Page 60]

RFC 6733 Diameter Base Protocol October 2012

The receiver only issues commands to its peers that have advertised support for the Diameter application that defines the command. A Diameter node MUST cache the supported Application Ids in order to

ensure that unrecognized commands and/or AVPs are not unnecessarily sent to a peer.

A receiver of a Capabilities-Exchange-Request (CER) message that does not have any applications in common with the sender MUST return a Capabilities-Exchange-Answer (CEA) with the Result-Code AVP set to DIAMETER_NO_COMMON_APPLICATION and SHOULD disconnect the transport layer connection. Note that receiving a CER or CEA from a peer advertising itself as a relay (see Section 2.4) MUST be interpreted as having common applications with the peer.

The receiver of the Capabilities-Exchange-Request (CER) MUST determine common applications by computing the intersection of its own set of supported Application Ids against all of the Application-Id AVPs (Auth-Application-Id, Acct-Application-Id, and Vendor-Specific-Application-Id) present in the CER. The value of the Vendor-Id AVP in the Vendor-Specific-Application-Id MUST NOT be used during computation. The sender of the Capabilities-Exchange-Answer (CEA) SHOULD include all of its supported applications as a hint to the receiver regarding all of its application capabilities.

Diameter implementations SHOULD first attempt to establish a TLS/TCP and DTLS/SCTP connection prior to the CER/CEA exchange. This protects the capabilities information of both peers. To support older Diameter implementations that do not fully conform to this document, the transport security MAY still be negotiated via an Inband-Security AVP. In this case, the receiver of a Capabilities-Exchange-Request (CER) message that does not have any security mechanisms in common with the sender MUST return a Capabilities-Exchange-Answer (CEA) with the Result-Code AVP set to DIAMETER_NO_COMMON_SECURITY and SHOULD disconnect the transport layer connection.

CERs received from unknown peers MAY be silently discarded, or a CEA MAY be issued with the Result-Code AVP set to DIAMETER_UNKNOWN_PEER. In both cases, the transport connection is closed. If the local policy permits receiving CERs from unknown hosts, a successful CEA MAY be returned. If a CER from an unknown peer is answered with a successful CEA, the lifetime of the peer entry is equal to the lifetime of the transport connection. In case of a transport failure, all the pending transactions destined to the unknown peer can be discarded.

The CER and CEA messages MUST NOT be proxied, redirected, or relayed.

Since the CER/CEA messages cannot be proxied, it is still possible that an upstream agent will receive a message for which it has no available peers to handle the application that corresponds to the Command Code. In such instances, the 'E' bit is set in the answer message (Section 7) with the Result-Code AVP set to DIAMETER_UNABLE_TO_DELIVER to inform the downstream agent to take action (e.g., re-routing request to an alternate peer).

With the exception of the Capabilities-Exchange-Request message, a message of type Request that includes the Auth-Application-Id or Acct-Application-Id AVPs, or a message with an application-specific Command Code MAY only be forwarded to a host that has explicitly

1.4 Standards Compliance

advertised support for the application (or has advertised the Relay Application Id).

5.3.1. Capabilities-Exchange-Request

The Capabilities-Exchange-Request (CER), indicated by the Command Code set to 257 and the Command Flags' 'R' bit set, is sent to exchange local capabilities. Upon detection of a transport failure, this message MUST NOT be sent to an alternate peer.

When Diameter is run over SCTP [RFC4960] or DTLS/SCTP [RFC6083], which allow for connections to span multiple interfaces and multiple IP addresses, the Capabilities-Exchange-Request message MUST contain one Host-IP-Address AVP for each potential IP address that MAY be locally used when transmitting Diameter messages.

Message Format

```
<CER> ::= < Diameter Header: 257, REQ >
  { Origin-Host }
  { Origin-Realm }
  1* { Host-IP-Address }
  { Vendor-Id }
  { Product-Name }
  [ Origin-State-Id ]
  * [ Supported-Vendor-Id ]
  * [ Auth-Application-Id ]
  * [ Inband-Security-Id ]
  * [ Acct-Application-Id ]
  * [ Vendor-Specific-Application-Id ]
  [ Firmware-Revision ]
  * [ AVP ]
```

Fajardo, et al.

Standards Track

[Page 62]

RFC 6733

Diameter Base Protocol

October 2012

5.3.2. Capabilities-Exchange-Answer

The Capabilities-Exchange-Answer (CEA), indicated by the Command Code set to 257 and the Command Flags' 'R' bit cleared, is sent in response to a CER message.

When Diameter is run over SCTP [RFC4960] or DTLS/SCTP [RFC6083], which allow connections to span multiple interfaces, hence, multiple IP addresses, the Capabilities-Exchange-Answer message MUST contain one Host-IP-Address AVP for each potential IP address that MAY be locally used when transmitting Diameter messages.

Message Format

```
<CEA> ::= < Diameter Header: 257 >
  { Result-Code }
  { Origin-Host }
  { Origin-Realm }
  1* { Host-IP-Address }
  { Vendor-Id }
  { Product-Name }
```

```

    [ Origin-State-Id ]
    [ Error-Message ]
    [ Failed-AVP ]
    * [ Supported-Vendor-Id ]
    * [ Auth-Application-Id ]
    * [ Inband-Security-Id ]
    * [ Acct-Application-Id ]
    * [ Vendor-Specific-Application-Id ]
    [ Firmware-Revision ]
    * [ AVP ]

```

5.3.3. Vendor-Id AVP

The Vendor-Id AVP (AVP Code 266) is of type Unsigned32 and contains the IANA "SMI Network Management Private Enterprise Codes" [ENTERPRISE] value assigned to the Diameter Software vendor. It is envisioned that the combination of the Vendor-Id, Product-Name (Section 5.3.7), and Firmware-Revision (Section 5.3.4) AVPs may provide useful debugging information.

A Vendor-Id value of zero in the CER or CEA message is reserved and indicates that this field is ignored.

Fajardo, et al.

Standards Track

[Page 63]

RFC 6733

Diameter Base Protocol

October 2012

5.3.4. Firmware-Revision AVP

The Firmware-Revision AVP (AVP Code 267) is of type Unsigned32 and is used to inform a Diameter peer of the firmware revision of the issuing device.

For devices that do not have a firmware revision (general-purpose computers running Diameter software modules, for instance), the revision of the Diameter software module may be reported instead.

5.3.5. Host-IP-Address AVP

The Host-IP-Address AVP (AVP Code 257) is of type Address and is used to inform a Diameter peer of the sender's IP address. All source addresses that a Diameter node expects to use with SCTP [RFC4960] or DTLS/SCTP [RFC6083] MUST be advertised in the CER and CEA messages by including a Host-IP-Address AVP for each address.

5.3.6. Supported-Vendor-Id AVP

The Supported-Vendor-Id AVP (AVP Code 265) is of type Unsigned32 and contains the IANA "SMI Network Management Private Enterprise Codes" [ENTERPRISE] value assigned to a vendor other than the device vendor but including the application vendor. This is used in the CER and CEA messages in order to inform the peer that the sender supports (a subset of) the Vendor-Specific AVPs defined by the vendor identified in this AVP. The value of this AVP MUST NOT be set to zero. Multiple instances of this AVP containing the same value SHOULD NOT be sent.

Fajardo, et al. Standards Track [Page 66]
RFC 6733 Diameter Base Protocol October 2012

5.5.1. Device-Watchdog-Request

The Device-Watchdog-Request (DWR), indicated by the Command Code set to 280 and the Command Flags' 'R' bit set, is sent to a peer when no traffic has been exchanged between two peers (see Section 5.5.3). Upon detection of a transport failure, this message MUST NOT be sent to an alternate peer.

Message Format

```
<DWR> ::= < Diameter Header: 280, REQ >
         { Origin-Host }
         { Origin-Realm }
         [ Origin-State-Id ]
         * [ AVP ]
```

5.5.2. Device-Watchdog-Answer

The Device-Watchdog-Answer (DWA), indicated by the Command Code set to 280 and the Command Flags' 'R' bit cleared, is sent as a response to the Device-Watchdog-Request message.

Message Format

```
<DWA> ::= < Diameter Header: 280 >
         { Result-Code }
         { Origin-Host }
         { Origin-Realm }
         [ Error-Message ]
         [ Failed-AVP ]
         [ Origin-State-Id ]
         * [ AVP ]
```

5.5.3. Transport Failure Algorithm

The transport failure algorithm is defined in [RFC3539]. All Diameter implementations MUST support the algorithm defined in that specification in order to be compliant to the Diameter base protocol.

5.5.4. Failover and Failback Procedures

In the event that a transport failure is detected with a peer, it is necessary for all pending request messages to be forwarded to an alternate agent, if possible. This is commonly referred to as "failover".

Fajardo, et al. Standards Track [Page 67]
RFC 6733 Diameter Base Protocol October 2012

In order for a Diameter node to perform failover procedures, it is

necessary for the node to maintain a pending message queue for a given peer. When an answer message is received, the corresponding request is removed from the queue. The Hop-by-Hop Identifier field is used to match the answer with the queued request.

When a transport failure is detected, if possible, all messages in the queue are sent to an alternate agent with the T flag set. On booting a Diameter client or agent, the T flag is also set on any remaining records in non-volatile storage that are still waiting to be transmitted. An example of a case where it is not possible to forward the message to an alternate server is when the message has a fixed destination, and the unavailable peer is the message's final destination (see Destination-Host AVP). Such an error requires that the agent return an answer message with the 'E' bit set and the Result-Code AVP set to DIAMETER_UNABLE_TO_DELIVER.

It is important to note that multiple identical requests or answers MAY be received as a result of a failover. The End-to-End Identifier field in the Diameter header along with the Origin-Host AVP MUST be used to identify duplicate messages.

As described in Section 2.1, a connection request should be periodically attempted with the failed peer in order to re-establish the transport connection. Once a connection has been successfully established, messages can once again be forwarded to the peer. This is commonly referred to as "failback".

5.6. Peer State Machine

This section contains a finite state machine that MUST be observed by all Diameter implementations. Each Diameter node MUST follow the state machine described below when communicating with each peer. Multiple actions are separated by commas, and may continue on succeeding lines, as space requires. Similarly, state and next state may also span multiple lines, as space requires.

This state machine is closely coupled with the state machine described in [RFC3539], which is used to open, close, failover, probe, and reopen transport connections. In particular, note that [RFC3539] requires the use of watchdog messages to probe connections. For Diameter, DWR and DWA messages are to be used.

The I- prefix is used to represent the initiator (connecting) connection, while the R- prefix is used to represent the responder (listening) connection. The lack of a prefix indicates that the event or action is the same regardless of the connection on which the event occurred.

Fajardo, et al. Standards Track [Page 68]

RFC 6733 Diameter Base Protocol October 2012

The stable states that a state machine may be in are Closed, I-Open, and R-Open; all other states are intermediate. Note that I-Open and R-Open are equivalent except for whether the initiator or responder transport connection is used for communication.

A CER message is always sent on the initiating connection immediately after the connection request is successfully completed. In the case of an election, one of the two connections will shut down. The responder connection will survive if the Origin-Host of the local Diameter entity is higher than that of the peer; the initiator

1.4 Standards Compliance

connection will survive if the peer's Origin-Host is higher. All subsequent messages are sent on the surviving connection. Note that the results of an election on one peer are guaranteed to be the inverse of the results on the other.

For TLS/TCP and DTLS/SCTP usage, a TLS/TCP and DTLS/SCTP handshake SHOULD begin when both ends are in the closed state prior to any Diameter message exchanges. The TLS/TCP and DTLS/SCTP connection SHOULD be established before sending any CER or CEA message to secure and protect the capabilities information of both peers. The TLS/TCP and DTLS/SCTP connection SHOULD be disconnected when the state machine moves to the closed state. When connecting to responders that do not conform to this document (i.e., older Diameter implementations that are not prepared to receive TLS/TCP and DTLS/SCTP connections in the closed state), the initial TLS/TCP and DTLS/SCTP connection attempt will fail. The initiator MAY then attempt to connect via TCP or SCTP and initiate the TLS/TCP and DTLS/SCTP handshake when both ends are in the open state. If the handshake is successful, all further messages will be sent via TLS/TCP and DTLS/SCTP. If the handshake fails, both ends move to the closed state.

The state machine constrains only the behavior of a Diameter implementation as seen by Diameter peers through events on the wire.

Any implementation that produces equivalent results is considered compliant.

Fajardo, et al. Standards Track [Page 69]
 RFC 6733 Diameter Base Protocol October 2012

state	event	action	next state
Closed	Start	I-Snd-Conn-Req	Wait-Conn-Ack
	R-Conn-CER	R-Accept, Process-CER, R-Snd-CEA	R-Open
Wait-Conn-Ack	I-Rcv-Conn-Ack	I-Snd-CER	Wait-I-CEA
	I-Rcv-Conn-Nack	Cleanup	Closed
	R-Conn-CER	R-Accept, Process-CER	Wait-Conn-Ack/ Elect
	Timeout	Error	Closed
Wait-I-CEA	I-Rcv-CEA	Process-CEA	I-Open
	R-Conn-CER	R-Accept, Process-CER, Elect	Wait>Returns
	I-Peer-Disc	I-Disc	Closed
	I-Rcv-Non-CEA	Error	Closed

	Timeout	Error	Closed
Wait-Conn-Ack/ Elect	I-Rcv-Conn-Ack	I-Snd-CER,Elect	Wait>Returns
	I-Rcv-Conn-Nack	R-Snd-CEA	R-Open
	R-Peer-Disc	R-Disc	Wait-Conn-Ack
	R-Conn-CER	R-Reject	Wait-Conn-Ack/ Elect
	Timeout	Error	Closed
Wait>Returns	Win-Election	I-Disc,R-Snd-CEA	R-Open
	I-Peer-Disc	I-Disc, R-Snd-CEA	R-Open
	I-Rcv-CEA	R-Disc	I-Open
	R-Peer-Disc	R-Disc	Wait-I-CEA
	R-Conn-CER	R-Reject	Wait>Returns
	Timeout	Error	Closed
R-Open	Send-Message	R-Snd-Message	R-Open
	R-Rcv-Message	Process	R-Open
	R-Rcv-DWR	Process-DWR, R-Snd-DWA	R-Open
	R-Rcv-DWA	Process-DWA	R-Open
	R-Conn-CER	R-Reject	R-Open
	Stop	R-Snd-DPR	Closing
	R-Rcv-DPR	R-Snd-DPA	Closing
	R-Peer-Disc	R-Disc	Closed

Fajardo, et al.

Standards Track

[Page 70]

RFC 6733

Diameter Base Protocol

October 2012

I-Open	Send-Message	I-Snd-Message	I-Open
	I-Rcv-Message	Process	I-Open
	I-Rcv-DWR	Process-DWR, I-Snd-DWA	I-Open
	I-Rcv-DWA	Process-DWA	I-Open
	R-Conn-CER	R-Reject	I-Open
	Stop	I-Snd-DPR	Closing
	I-Rcv-DPR	I-Snd-DPA	Closing
	I-Peer-Disc	I-Disc	Closed
Closing	I-Rcv-DPA	I-Disc	Closed
	R-Rcv-DPA	R-Disc	Closed
	Timeout	Error	Closed
	I-Peer-Disc	I-Disc	Closed
	R-Peer-Disc	R-Disc	Closed

5.6.1. Incoming Connections

When a connection request is received from a Diameter peer, it is not, in the general case, possible to know the identity of that peer until a CER is received from it. This is because host and port determine the identity of a Diameter peer; the source port of an incoming connection is arbitrary. Upon receipt of a CER, the identity of the connecting peer can be uniquely determined from the Origin-Host.

For this reason, a Diameter peer must employ logic separate from the state machine to receive connection requests, accept them, and await

1.4 Standards Compliance

the CER. Once the CER arrives on a new connection, the Origin-Host that identifies the peer is used to locate the state machine associated with that peer, and the new connection and CER are passed to the state machine as an R-Conn-CER event.

The logic that handles incoming connections SHOULD close and discard the connection if any message other than a CER arrives or if an implementation-defined timeout occurs prior to receipt of CER.

Because handling of incoming connections up to and including receipt of a CER requires logic, separate from that of any individual state machine associated with a particular peer, it is described separately in this section rather than in the state machine above.

5.6.2. Events

Transitions and actions in the automaton are caused by events. In this section, we will ignore the I- and R- prefixes, since the actual event would be identical, but it would occur on one of two possible connections.

Fajardo, et al. Standards Track [Page 71]

RFC 6733 Diameter Base Protocol October 2012

Start	The Diameter application has signaled that a connection should be initiated with the peer.
R-Conn-CER	An acknowledgement is received stating that the transport connection has been established, and the associated CER has arrived.
Rcv-Conn-Ack	A positive acknowledgement is received confirming that the transport connection is established.
Rcv-Conn-Nack	A negative acknowledgement was received stating that the transport connection was not established.
Timeout	An application-defined timer has expired while waiting for some event.
Rcv-CER	A CER message from the peer was received.
Rcv-CEA	A CEA message from the peer was received.
Rcv-Non-CEA	A message, other than a CEA, from the peer was received.
Peer-Disc	A disconnection indication from the peer was received.
Rcv-DPR	A DPR message from the peer was received.
Rcv-DPA	A DPA message from the peer was received.
Win-Election	An election was held, and the local node was the winner.
Send-Message	A message is to be sent.
Rcv-Message	A message other than CER, CEA, DPR, DPA, DWR, or DWA was received.

Fajardo, et al. Standards Track [Page 73]
RFC 6733 Diameter Base Protocol October 2012

5.6.4. The Election Process

The election is performed on the responder. The responder compares the Origin-Host received in the CER with its own Origin-Host as two streams of octets. If the local Origin-Host lexicographically succeeds the received Origin-Host, a Win-Election event is issued locally. Diameter identities are in ASCII form; therefore, the lexical comparison is consistent with DNS case insensitivity, where octets that fall in the ASCII range 'a' through 'z' MUST compare equally to their uppercase counterparts between 'A' and 'Z'. See Appendix D for interactions between the Diameter protocol and Internationalized Domain Name (IDNs).

The winner of the election MUST close the connection it initiated. Historically, maintaining the responder side of a connection was more efficient than maintaining the initiator side. However, current practices makes this distinction irrelevant.

6. Diameter Message Processing

This section describes how Diameter requests and answers are created and processed.

6.1. Diameter Request Routing Overview

A request is sent towards its final destination using one of the following three combinations of the Destination-Realm and Destination-Host AVPs:

- o A request that is not able to be proxied (such as a CER) MUST NOT contain either Destination-Realm or Destination-Host AVPs.
- o A request that needs to be sent to a home server serving a specific realm, but not to a specific server (such as the first request of a series of round trips), MUST contain a Destination-Realm AVP but MUST NOT contain a Destination-Host AVP. For Diameter clients, the value of the Destination-Realm AVP MAY be extracted from the User-Name AVP, or other methods.
- o Otherwise, a request that needs to be sent to a specific home server among those serving a given realm MUST contain both the Destination-Realm and Destination-Host AVPs.

The Destination-Host AVP is used as described above when the destination of the request is fixed, which includes:

- o Authentication requests that span multiple round trips.

Fajardo, et al. Standards Track [Page 74]
RFC 6733 Diameter Base Protocol October 2012

1.4 Standards Compliance

and

- o the Destination-Host and Destination-Realm AVPs MUST be set to the appropriate values, as described in Section 6.1.

6.1.2. Sending a Request

When sending a request, originated either locally or as the result of a forwarding or routing operation, the following procedures SHOULD be followed:

- o The Hop-by-Hop Identifier SHOULD be set to a locally unique value.
- o The message SHOULD be saved in the list of pending requests.

Other actions to perform on the message based on the particular role the agent is playing are described in the following sections.

6.1.3. Receiving Requests

A relay or proxy agent MUST check for forwarding loops when receiving requests. A loop is detected if the server finds its own identity in a Route-Record AVP. When such an event occurs, the agent MUST answer with the Result-Code AVP set to DIAMETER_LOOP_DETECTED.

6.1.4. Processing Local Requests

A request is known to be for local consumption when one of the following conditions occurs:

- o The Destination-Host AVP contains the local host's identity;
- o The Destination-Host AVP is not present, the Destination-Realm AVP contains a realm the server is configured to process locally, and the Diameter application is locally supported; or
- o Both the Destination-Host and the Destination-Realm are not present.

Fajardo, et al.

Standards Track

[Page 76]

RFC 6733

Diameter Base Protocol

October 2012

When a request is locally processed, the rules in Section 6.2 should be used to generate the corresponding answer.

6.1.5. Request Forwarding

Request forwarding is done using the Diameter peer table. The Diameter peer table contains all of the peers with which the local node is able to directly communicate.

When a request is received, and the host encoded in the Destination-Host AVP is one that is present in the peer table, the message SHOULD be forwarded to the peer.

6.1.6. Request Routing

Diameter request message routing is done via realms and Application Ids. A Diameter message that may be forwarded by Diameter agents

(proxies, redirect agents, or relay agents) MUST include the target realm in the Destination-Realm AVP. Request routing SHOULD rely on the Destination-Realm AVP and the Application Id present in the request message header to aid in the routing decision. The realm MAY be retrieved from the User-Name AVP, which is in the form of a Network Access Identifier (NAI). The realm portion of the NAI is inserted in the Destination-Realm AVP.

Diameter agents MAY have a list of locally supported realms and applications, and they MAY have a list of externally supported realms and applications. When a request is received that includes a realm and/or application that is not locally supported, the message is routed to the peer configured in the routing table (see Section 2.7).

Realm names and Application Ids are the minimum supported routing criteria, additional information may be needed to support redirect semantics.

6.1.7. Predictive Loop Avoidance

Before forwarding or routing a request, Diameter agents, in addition to performing the processing described in Section 6.1.3, SHOULD check for the presence of a candidate route's peer identity in any of the Route-Record AVPs. In the event of the agent detecting the presence of a candidate route's peer identity in a Route-Record AVP, the agent MUST ignore such a route for the Diameter request message and attempt alternate routes if any exist. In case all the candidate routes are eliminated by the above criteria, the agent SHOULD return a DIAMETER_UNABLE_TO_DELIVER message.

Fajardo, et al.

Standards Track

[Page 77]

RFC 6733

Diameter Base Protocol

October 2012

6.1.8. Redirecting Requests

When a redirect agent receives a request whose routing entry is set to REDIRECT, it MUST reply with an answer message with the 'E' bit set, while maintaining the Hop-by-Hop Identifier in the header, and include the Result-Code AVP to DIAMETER_REDIRECT_INDICATION. Each of the servers associated with the routing entry are added in a separate Redirect-Host AVP.

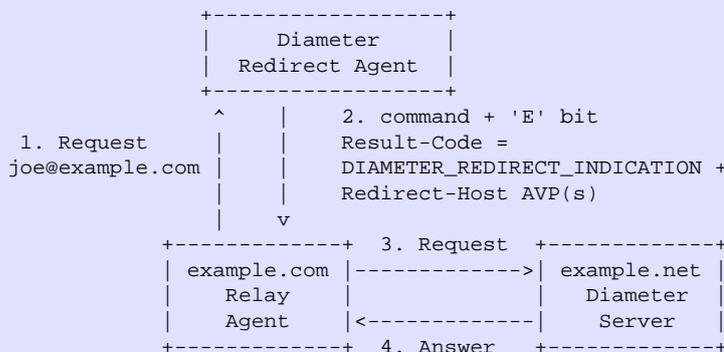


Figure 5: Diameter Redirect Agent

least 128 bits in strength.

- o The keys should not be used for any other purpose than generating and verifying instances of the Proxy-Info AVP.
- o The keys should be changed regularly.
- o The keys should be changed if the AVP format or cryptographic protection algorithms change.

The message is then forwarded to the next hop, as identified in the routing table.

Fajardo, et al.

Standards Track

[Page 79]

RFC 6733

Diameter Base Protocol

October 2012

Figure 6 provides an example of message routing using the procedures listed in these sections.

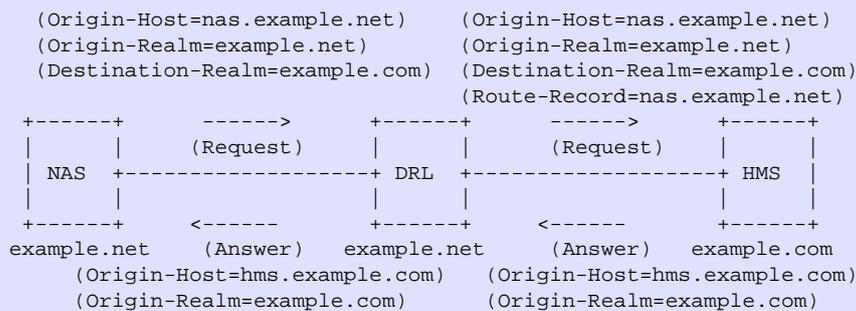


Figure 6: Routing of Diameter messages

Relay and proxy agents are not required to perform full inspection of incoming messages. At a minimum, validation of the message header and relevant routing AVPs has to be done when relaying messages. Proxy agents may optionally perform more in-depth message validation for applications in which it is interested.

6.2. Diameter Answer Processing

When a request is locally processed, the following procedures **MUST** be applied to create the associated answer, in addition to any additional procedures that **MAY** be discussed in the Diameter application defining the command:

- o The same Hop-by-Hop Identifier in the request is used in the answer.
- o The local host's identity is encoded in the Origin-Host AVP.
- o The Destination-Host and Destination-Realm AVPs **MUST NOT** be present in the answer message.
- o The Result-Code AVP is added with its value indicating success or failure.
- o If the Session-Id is present in the request, it **MUST** be included in the answer.

- o Any Proxy-Info AVPs in the request MUST be added to the answer message, in the same order they were present in the request.

Fajardo, et al. Standards Track [Page 80]

RFC 6733 Diameter Base Protocol October 2012

- o The 'P' bit is set to the same value as the one in the request.
- o The same End-to-End identifier in the request is used in the answer.

Note that the error messages (see Section 7) are also subjected to the above processing rules.

6.2.1. Processing Received Answers

A Diameter client or proxy MUST match the Hop-by-Hop Identifier in an answer received against the list of pending requests. The corresponding message should be removed from the list of pending requests. It SHOULD ignore answers received that do not match a known Hop-by-Hop Identifier.

6.2.2. Relaying and Proxying Answers

If the answer is for a request that was proxied or relayed, the agent MUST restore the original value of the Diameter header's Hop-by-Hop Identifier field.

If the last Proxy-Info AVP in the message is targeted to the local Diameter server, the AVP MUST be removed before the answer is forwarded.

If a relay or proxy agent receives an answer with a Result-Code AVP indicating a failure, it MUST NOT modify the contents of the AVP. Any additional local errors detected SHOULD be logged but not reflected in the Result-Code AVP. If the agent receives an answer message with a Result-Code AVP indicating success, and it wishes to modify the AVP to indicate an error, it MUST modify the Result-Code AVP to contain the appropriate error in the message destined towards the access device as well as include the Error-Reporting-Host AVP; it MUST also issue an STR on behalf of the access device towards the Diameter server.

The agent MUST then send the answer to the host that it received the original request from.

6.3. Origin-Host AVP

The Origin-Host AVP (AVP Code 264) is of type DiameterIdentity, and it MUST be present in all Diameter messages. This AVP identifies the endpoint that originated the Diameter message. Relay agents MUST NOT modify this AVP.

Fajardo, et al. Standards Track [Page 81]

The value of the Origin-Host AVP is guaranteed to be unique within a single host.

Note that the Origin-Host AVP may resolve to more than one address as the Diameter peer may support more than one address.

This AVP SHOULD be placed as close to the Diameter header as possible.

6.4. Origin-Realm AVP

The Origin-Realm AVP (AVP Code 296) is of type DiameterIdentity. This AVP contains the Realm of the originator of any Diameter message and MUST be present in all messages.

This AVP SHOULD be placed as close to the Diameter header as possible.

6.5. Destination-Host AVP

The Destination-Host AVP (AVP Code 293) is of type DiameterIdentity. This AVP MUST be present in all unsolicited agent initiated messages, MAY be present in request messages, and MUST NOT be present in answer messages.

The absence of the Destination-Host AVP will cause a message to be sent to any Diameter server supporting the application within the realm specified in Destination-Realm AVP.

This AVP SHOULD be placed as close to the Diameter header as possible.

6.6. Destination-Realm AVP

The Destination-Realm AVP (AVP Code 283) is of type DiameterIdentity and contains the realm to which the message is to be routed. The Destination-Realm AVP MUST NOT be present in answer messages. Diameter clients insert the realm portion of the User-Name AVP. Diameter servers initiating a request message use the value of the Origin-Realm AVP from a previous message received from the intended target host (unless it is known a priori). When present, the Destination-Realm AVP is used to perform message routing decisions.

The CCF for a request message that includes the Destination-Realm AVP SHOULD list the Destination-Realm AVP as a required AVP (an AVP indicated as {AVP}); otherwise, the message is inherently a non-routable message.

This AVP SHOULD be placed as close to the Diameter header as possible.

6.7. Routing AVPs

The AVPs defined in this section are Diameter AVPs used for routing

1.4 Standards Compliance

purposes. These AVPs change as Diameter messages are processed by agents.

6.7.1. Route-Record AVP

The Route-Record AVP (AVP Code 282) is of type DiameterIdentity. The identity added in this AVP MUST be the same as the one received in the Origin-Host of the Capabilities Exchange message.

6.7.2. Proxy-Info AVP

The Proxy-Info AVP (AVP Code 284) is of type Grouped. This AVP contains the identity and local state information of the Diameter node that creates and adds it to a message. The Grouped Data field has the following CCF grammar:

```
Proxy-Info ::= < AVP Header: 284 >
              { Proxy-Host }
              { Proxy-State }
              * [ AVP ]
```

6.7.3. Proxy-Host AVP

The Proxy-Host AVP (AVP Code 280) is of type DiameterIdentity. This AVP contains the identity of the host that added the Proxy-Info AVP.

6.7.4. Proxy-State AVP

The Proxy-State AVP (AVP Code 33) is of type OctetString. It contains state information that would otherwise be stored at the Diameter entity that created it. As such, this AVP MUST be treated as opaque data by other Diameter entities.

6.8. Auth-Application-Id AVP

The Auth-Application-Id AVP (AVP Code 258) is of type Unsigned32 and is used in order to advertise support of the Authentication and Authorization portion of an application (see Section 2.4). If present in a message other than CER and CEA, the value of the Auth-Application-Id AVP MUST match the Application Id present in the Diameter message header.

6.9. Acct-Application-Id AVP

The Acct-Application-Id AVP (AVP Code 259) is of type Unsigned32 and is used in order to advertise support of the accounting portion of an application (see Section 2.4). If present in a message other than CER and CEA, the value of the Acct-Application-Id AVP MUST match the Application Id present in the Diameter message header.

6.10. Inband-Security-Id AVP

The Inband-Security-Id AVP (AVP Code 299) is of type Unsigned32 and is used in order to advertise support of the security portion of the application. The use of this AVP in CER and CEA messages is NOT RECOMMENDED. Instead, discovery of a Diameter entity's security capabilities can be done either through static configuration or via

Diameter Peer Discovery as described in Section 5.2.

The following values are supported:

NO_INBAND_SECURITY 0

This peer does not support TLS/TCP and DTLS/SCTP. This is the default value, if the AVP is omitted.

TLS 1

This node supports TLS/TCP [RFC5246] and DTLS/SCTP [RFC6083] security.

6.11. Vendor-Specific-Application-Id AVP

The Vendor-Specific-Application-Id AVP (AVP Code 260) is of type Grouped and is used to advertise support of a vendor-specific Diameter application. Exactly one instance of either Auth-Application-Id or Acct-Application-Id AVP MUST be present. The Application Id carried by either Auth-Application-Id or Acct-Application-Id AVP MUST comply with vendor-specific Application Id assignment described in Section 11.3. It MUST also match the Application Id present in the Diameter header except when used in a CER or CEA message.

The Vendor-Id AVP is an informational AVP pertaining to the vendor who may have authorship of the vendor-specific Diameter application. It MUST NOT be used as a means of defining a completely separate vendor-specific Application Id space.

Fajardo, et al.	Standards Track	[Page 84]
RFC 6733	Diameter Base Protocol	October 2012

The Vendor-Specific-Application-Id AVP SHOULD be placed as close to the Diameter header as possible.

AVP Format

```
<Vendor-Specific-Application-Id> ::= < AVP Header: 260 >
    { Vendor-Id }
    [ Auth-Application-Id ]
    [ Acct-Application-Id ]
```

A Vendor-Specific-Application-Id AVP MUST contain exactly one of either Auth-Application-Id or Acct-Application-Id. If a Vendor-Specific-Application-Id is received without one of these two AVPs, then the recipient SHOULD issue an answer with a Result-Code set to DIAMETER_MISSING_AVP. The answer SHOULD also include a Failed-AVP, which MUST contain an example of an Auth-Application-Id AVP and an Acct-Application-Id AVP.

If a Vendor-Specific-Application-Id is received that contains both Auth-Application-Id and Acct-Application-Id, then the recipient MUST issue an answer with Result-Code set to DIAMETER_AVP_OCCURS_TOO_MANY_TIMES. The answer MUST also include a Failed-AVP, which MUST contain the received Auth-Application-Id AVP and Acct-Application-Id AVP.

1.4 Standards Compliance

6.12. Redirect-Host AVP

The Redirect-Host AVP (AVP Code 292) is of type DiameterURI. One or more instances of this AVP MUST be present if the answer message's 'E' bit is set and the Result-Code AVP is set to DIAMETER_REDIRECT_INDICATION.

Upon receiving the above, the receiving Diameter node SHOULD forward the request directly to one of the hosts identified in these AVPs. The server contained in the selected Redirect-Host AVP SHOULD be used for all messages matching the criteria set by the Redirect-Host-Usage AVP.

6.13. Redirect-Host-Usage AVP

The Redirect-Host-Usage AVP (AVP Code 261) is of type Enumerated. This AVP MAY be present in answer messages whose 'E' bit is set and the Result-Code AVP is set to DIAMETER_REDIRECT_INDICATION.

When present, this AVP provides hints about how the routing entry resulting from the Redirect-Host is to be used. The following values are supported:

Fajardo, et al. Standards Track [Page 85]
RFC 6733 Diameter Base Protocol October 2012

DONT_CACHE 0

The host specified in the Redirect-Host AVP SHOULD NOT be cached. This is the default value.

ALL_SESSION 1

All messages within the same session, as defined by the same value of the Session-ID AVP SHOULD be sent to the host specified in the Redirect-Host AVP.

ALL_REALM 2

All messages destined for the realm requested SHOULD be sent to the host specified in the Redirect-Host AVP.

REALM_AND_APPLICATION 3

All messages for the application requested to the realm specified SHOULD be sent to the host specified in the Redirect-Host AVP.

ALL_APPLICATION 4

All messages for the application requested SHOULD be sent to the host specified in the Redirect-Host AVP.

ALL_HOST 5

All messages that would be sent to the host that generated the Redirect-Host SHOULD be sent to the host specified in the Redirect-Host AVP.

ALL_USER 6

All messages for the user requested SHOULD be sent to the host specified in the Redirect-Host AVP.

When multiple cached routes are created by redirect indications and they differ only in redirect usage and peers to forward requests to (see Section 6.1.8), a precedence rule MUST be applied to the redirect usage values of the cached routes during normal routing to resolve contentions that may occur. The precedence rule is the order that dictate which redirect usage should be considered before any other as they appear. The order is as follows:

Fajardo, et al. Standards Track [Page 86]
RFC 6733 Diameter Base Protocol October 2012

1. ALL_SESSION
2. ALL_USER
3. REALM_AND_APPLICATION
4. ALL_REALM
5. ALL_APPLICATION
6. ALL_HOST

6.14. Redirect-Max-Cache-Time AVP

The Redirect-Max-Cache-Time AVP (AVP Code 262) is of type Unsigned32. This AVP MUST be present in answer messages whose 'E' bit is set, whose Result-Code AVP is set to DIAMETER_REDIRECT_INDICATION, and whose Redirect-Host-Usage AVP set to a non-zero value.

This AVP contains the maximum number of seconds the peer and route table entries, created as a result of the Redirect-Host, SHOULD be cached. Note that once a host is no longer reachable, any associated cache, peer, and routing table entries MUST be deleted.

7. Error Handling

There are two different types of errors in Diameter; protocol errors and application errors. A protocol error is one that occurs at the base protocol level and MAY require per-hop attention (e.g., a message routing error). Application errors, on the other hand, generally occur due to a problem with a function specified in a Diameter application (e.g., user authentication, missing AVP).

Result-Code AVP values that are used to report protocol errors MUST only be present in answer messages whose 'E' bit is set. When a request message is received that causes a protocol error, an answer message is returned with the 'E' bit set, and the Result-Code AVP is set to the appropriate protocol error value. As the answer is sent back towards the originator of the request, each proxy or relay agent MAY take action on the message.

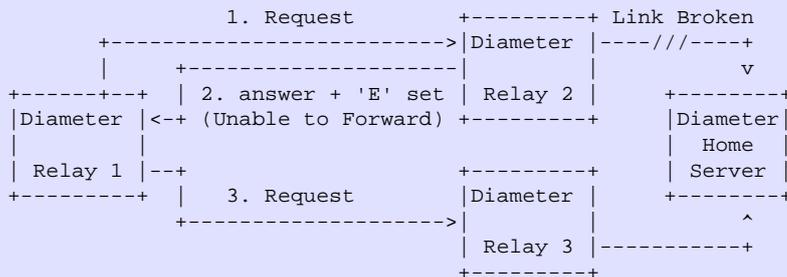


Figure 7: Example of Protocol Error Causing Answer Message

Figure 7 provides an example of a message forwarded upstream by a Diameter relay. When the message is received by Relay 2, and it detects that it cannot forward the request to the home server, an answer message is returned with the 'E' bit set and the Result-Code AVP set to DIAMETER_UNABLE_TO_DELIVER. Given that this error falls within the protocol error category, Relay 1 would take special action, and given the error, attempt to route the message through its alternate Relay 3.

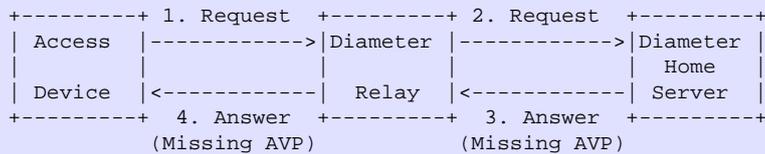


Figure 8: Example of Application Error Answer Message

Figure 8 provides an example of a Diameter message that caused an application error. When application errors occur, the Diameter entity reporting the error clears the 'R' bit in the Command Flags and adds the Result-Code AVP with the proper value. Application errors do not require any proxy or relay agent involvement; therefore, the message would be forwarded back to the originator of the request.

In the case where the answer message itself contains errors, any related session SHOULD be terminated by sending an STR or ASR message. The Termination-Cause AVP in the STR MAY be filled with the appropriate value to indicate the cause of the error. An application MAY also send an application-specific request instead of an STR or ASR message to signal the error in the case where no state is maintained or to allow for some form of error recovery with the corresponding Diameter entity.

Fajardo, et al. Standards Track [Page 88]

RFC 6733 Diameter Base Protocol October 2012

There are certain Result-Code AVP application errors that require additional AVPs to be present in the answer. In these cases, the Diameter node that sets the Result-Code AVP to indicate the error MUST add the AVPs. Examples are as follows:

- o A request with an unrecognized AVP is received with the 'M' bit (Mandatory bit) set causes an answer to be sent with the Result-Code AVP set to `DIAMETER_AVP_UNSUPPORTED` and the Failed-AVP AVP containing the offending AVP.
- o A request with an AVP that is received with an unrecognized value causes an answer to be returned with the Result-Code AVP set to `DIAMETER_INVALID_AVP_VALUE`, with the Failed-AVP AVP containing the AVP causing the error.
- o A received command that is missing AVPs that are defined as required in the commands CCF; examples are AVPs indicated as {AVP}. The receiver issues an answer with the Result-Code set to `DIAMETER_MISSING_AVP` and creates an AVP with the AVP Code and other fields set as expected in the missing AVP. The created AVP is then added to the Failed-AVP AVP.

The Result-Code AVP describes the error that the Diameter node encountered in its processing. In case there are multiple errors, the Diameter node MUST report only the first error it encountered (detected possibly in some implementation-dependent order). The specific errors that can be described by this AVP are described in the following section.

7.1. Result-Code AVP

The Result-Code AVP (AVP Code 268) is of type Unsigned32 and indicates whether a particular request was completed successfully or an error occurred. All Diameter answer messages in IETF-defined Diameter application specifications MUST include one Result-Code AVP. A non-successful Result-Code AVP (one containing a non-2xxx value other than `DIAMETER_REDIRECT_INDICATION`) MUST include the Error-Reporting-Host AVP if the host setting the Result-Code AVP is different from the identity encoded in the Origin-Host AVP.

The Result-Code data field contains an IANA-managed 32-bit address space representing errors (see Section 11.3.2). Diameter provides the following classes of errors, all identified by the thousands digit in the decimal notation:

Fajardo, et al. Standards Track [Page 89]

RFC 6733 Diameter Base Protocol October 2012

- o 1xxx (Informational)
- o 2xxx (Success)

1.4 Standards Compliance

- o 3xxx (Protocol Errors)
- o 4xxx (Transient Failures)
- o 5xxx (Permanent Failure)

An unrecognized class (one whose first digit is not defined in this section) MUST be handled as a permanent failure.

7.1.1. Informational

Errors that fall within this category are used to inform the requester that a request could not be satisfied, and additional action is required on its part before access is granted.

DIAMETER_MULTI_ROUND_AUTH 1001

This informational error is returned by a Diameter server to inform the access device that the authentication mechanism being used requires multiple round trips, and a subsequent request needs to be issued in order for access to be granted.

7.1.2. Success

Errors that fall within the Success category are used to inform a peer that a request has been successfully completed.

DIAMETER_SUCCESS 2001

The request was successfully completed.

DIAMETER_LIMITED_SUCCESS 2002

When returned, the request was successfully completed, but additional processing is required by the application in order to provide service to the user.

7.1.3. Protocol Errors

Errors that fall within the Protocol Error category SHOULD be treated on a per-hop basis, and Diameter proxies MAY attempt to correct the error, if it is possible. Note that these errors MUST only be used in answer messages whose 'E' bit is set.

Fajardo, et al. Standards Track [Page 90]

RFC 6733 Diameter Base Protocol October 2012

DIAMETER_COMMAND_UNSUPPORTED 3001

This error code is used when a Diameter entity receives a message with a Command Code that it does not support.

DIAMETER_UNABLE_TO_DELIVER 3002

This error is given when Diameter cannot deliver the message to the destination, either because no host within the realm supporting the required application was available to process the request or because the Destination-Host AVP was given without the associated Destination-Realm AVP.

DIAMETER_REALM_NOT_SERVED 3003

The intended realm of the request is not recognized.

DIAMETER_TOO_BUSY 3004

When returned, a Diameter node SHOULD attempt to send the message to an alternate peer. This error MUST only be used when a specific server is requested, and it cannot provide the requested service.

DIAMETER_LOOP_DETECTED 3005

An agent detected a loop while trying to get the message to the intended recipient. The message MAY be sent to an alternate peer, if one is available, but the peer reporting the error has identified a configuration problem.

DIAMETER_REDIRECT_INDICATION 3006

A redirect agent has determined that the request could not be satisfied locally, and the initiator of the request SHOULD direct the request directly to the server, whose contact information has been added to the response. When set, the Redirect-Host AVP MUST be present.

DIAMETER_APPLICATION_UNSUPPORTED 3007

A request was sent for an application that is not supported.

DIAMETER_INVALID_HDR_BITS 3008

A request was received whose bits in the Diameter header were set either to an invalid combination or to a value that is inconsistent with the Command Code's definition.

Fajardo, et al.

Standards Track

[Page 91]

RFC 6733

Diameter Base Protocol

October 2012

DIAMETER_INVALID_AVP_BITS 3009

A request was received that included an AVP whose flag bits are set to an unrecognized value or that is inconsistent with the AVP's definition.

DIAMETER_UNKNOWN_PEER 3010

A CER was received from an unknown peer.

7.1.4. Transient Failures

Errors that fall within the transient failures category are used to inform a peer that the request could not be satisfied at the time it was received but MAY be able to satisfy the request in the future. Note that these errors MUST be used in answer messages whose 'E' bit is not set.

DIAMETER_AUTHENTICATION_REJECTED 4001

The authentication process for the user failed, most likely due to an invalid password used by the user. Further attempts MUST only

1.4 Standards Compliance

be tried after prompting the user for a new password.

DIAMETER_OUT_OF_SPACE 4002

A Diameter node received the accounting request but was unable to commit it to stable storage due to a temporary lack of space.

ELECTION_LOST 4003

The peer has determined that it has lost the election process and has therefore disconnected the transport connection.

7.1.5. Permanent Failures

Errors that fall within the permanent failures category are used to inform the peer that the request failed and should not be attempted again. Note that these errors SHOULD be used in answer messages whose 'E' bit is not set. In error conditions where it is not possible or efficient to compose application-specific answer grammar, answer messages with the 'E' bit set and which comply to the grammar described in Section 7.2 MAY also be used for permanent errors.

Fajardo, et al.

Standards Track

[Page 92]

RFC 6733

Diameter Base Protocol

October 2012

DIAMETER_AVP_UNSUPPORTED 5001

The peer received a message that contained an AVP that is not recognized or supported and was marked with the 'M' (Mandatory) bit. A Diameter message with this error MUST contain one or more Failed-AVP AVPs containing the AVPs that caused the failure.

DIAMETER_UNKNOWN_SESSION_ID 5002

The request contained an unknown Session-Id.

DIAMETER_AUTHORIZATION_REJECTED 5003

A request was received for which the user could not be authorized. This error could occur if the service requested is not permitted to the user.

DIAMETER_INVALID_AVP_VALUE 5004

The request contained an AVP with an invalid value in its data portion. A Diameter message indicating this error MUST include the offending AVPs within a Failed-AVP AVP.

DIAMETER_MISSING_AVP 5005

The request did not contain an AVP that is required by the Command Code definition. If this value is sent in the Result-Code AVP, a Failed-AVP AVP SHOULD be included in the message. The Failed-AVP AVP MUST contain an example of the missing AVP complete with the Vendor-Id if applicable. The value field of the missing AVP should be of correct minimum length and contain zeroes.

DIAMETER_RESOURCES_EXCEEDED 5006

A request was received that cannot be authorized because the user has already expended allowed resources. An example of this error condition is when a user that is restricted to one dial-up PPP port attempts to establish a second PPP connection.

DIAMETER_CONTRADICTING_AVPS 5007

The Home Diameter server has detected AVPs in the request that contradicted each other, and it is not willing to provide service to the user. The Failed-AVP AVP MUST be present, which contain the AVPs that contradicted each other.

Fajardo, et al. Standards Track [Page 93]

RFC 6733 Diameter Base Protocol October 2012

DIAMETER_AVP_NOT_ALLOWED 5008

A message was received with an AVP that MUST NOT be present. The Failed-AVP AVP MUST be included and contain a copy of the offending AVP.

DIAMETER_AVP_OCCURS_TOO_MANY_TIMES 5009

A message was received that included an AVP that appeared more often than permitted in the message definition. The Failed-AVP AVP MUST be included and contain a copy of the first instance of the offending AVP that exceeded the maximum number of occurrences.

DIAMETER_NO_COMMON_APPLICATION 5010

This error is returned by a Diameter node that receives a CER whereby no applications are common between the CER sending peer and the CER receiving peer.

DIAMETER_UNSUPPORTED_VERSION 5011

This error is returned when a request was received, whose version number is unsupported.

DIAMETER_UNABLE_TO_COMPLY 5012

This error is returned when a request is rejected for unspecified reasons.

DIAMETER_INVALID_BIT_IN_HEADER 5013

This error is returned when a reserved bit in the Diameter header is set to one (1) or the bits in the Diameter header are set incorrectly.

DIAMETER_INVALID_AVP_LENGTH 5014

The request contained an AVP with an invalid length. A Diameter message indicating this error MUST include the offending AVPs within a Failed-AVP AVP. In cases where the erroneous AVP length

Fajardo, et al. Standards Track [Page 95]

RFC 6733 Diameter Base Protocol October 2012

Note that the code used in the header is the same than the one found in the request message, but with the 'R' bit cleared and the 'E' bit set. The 'P' bit in the header is set to the same value as the one found in the request message.

7.3. Error-Message AVP

The Error-Message AVP (AVP Code 281) is of type UTF8String. It MAY accompany a Result-Code AVP as a human-readable error message. The Error-Message AVP is not intended to be useful in an environment where error messages are processed automatically. It SHOULD NOT be expected that the content of this AVP be parsed by network entities.

7.4. Error-Reporting-Host AVP

The Error-Reporting-Host AVP (AVP Code 294) is of type DiameterIdentity. This AVP contains the identity of the Diameter host that sent the Result-Code AVP to a value other than 2001 (Success), only if the host setting the Result-Code is different from the one encoded in the Origin-Host AVP. This AVP is intended to be used for troubleshooting purposes, and it MUST be set when the Result-Code AVP indicates a failure.

7.5. Failed-AVP AVP

The Failed-AVP AVP (AVP Code 279) is of type Grouped and provides debugging information in cases where a request is rejected or not fully processed due to erroneous information in a specific AVP. The value of the Result-Code AVP will provide information on the reason for the Failed-AVP AVP. A Diameter answer message SHOULD contain an instance of the Failed-AVP AVP that corresponds to the error indicated by the Result-Code AVP. For practical purposes, this Failed-AVP would typically refer to the first AVP processing error that a Diameter node encounters.

The possible reasons for this AVP are the presence of an improperly constructed AVP, an unsupported or unrecognized AVP, an invalid AVP value, the omission of a required AVP, the presence of an explicitly excluded AVP (see tables in Section 10) or the presence of two or more occurrences of an AVP that is restricted to 0, 1, or 0-1 occurrences.

A Diameter message SHOULD contain one Failed-AVP AVP, containing the entire AVP that could not be processed successfully. If the failure reason is omission of a required AVP, an AVP with the missing AVP code, the missing Vendor-Id, and a zero-filled payload of the minimum required length for the omitted AVP will be added. If the failure reason is an invalid AVP length where the reported length is less

Fajardo, et al. Standards Track [Page 96]

RFC 6733 Diameter Base Protocol October 2012

than the minimum AVP header length or greater than the reported message length, a copy of the offending AVP header and a zero-filled

1.4 Standards Compliance

payload of the minimum required length SHOULD be added.

In the case where the offending AVP is embedded within a Grouped AVP, the Failed-AVP MAY contain the grouped AVP, which in turn contains the single offending AVP. The same method MAY be employed if the grouped AVP itself is embedded in yet another grouped AVP and so on. In this case, the Failed-AVP MAY contain the grouped AVP hierarchy up to the single offending AVP. This enables the recipient to detect the location of the offending AVP when embedded in a group.

AVP Format

```
<Failed-AVP> ::= < AVP Header: 279 >
                1* {AVP}
```

7.6. Experimental-Result AVP

The Experimental-Result AVP (AVP Code 297) is of type Grouped, and indicates whether a particular vendor-specific request was completed successfully or whether an error occurred. This AVP has the following structure:

AVP Format

```
Experimental-Result ::= < AVP Header: 297 >
                       { Vendor-Id }
                       { Experimental-Result-Code }
```

The Vendor-Id AVP (see Section 5.3.3) in this grouped AVP identifies the vendor responsible for the assignment of the result code that follows. All Diameter answer messages defined in vendor-specific applications MUST include either one Result-Code AVP or one Experimental-Result AVP.

7.7. Experimental-Result-Code AVP

The Experimental-Result-Code AVP (AVP Code 298) is of type Unsigned32 and contains a vendor-assigned value representing the result of processing the request.

It is recommended that vendor-specific result codes follow the same conventions given for the Result-Code AVP regarding the different types of result codes and the handling of errors (for non-2xxx values).

8. Diameter User Sessions

In general, Diameter can provide two different types of services to applications. The first involves authentication and authorization, and it can optionally make use of accounting. The second only makes use of accounting.

When a service makes use of the authentication and/or authorization portion of an application, and a user requests access to the network, the Diameter client issues an auth request to its local server. The auth request is defined in a service-specific Diameter application

(e.g., NASREQ). The request contains a Session-Id AVP, which is used in subsequent messages (e.g., subsequent authorization, accounting, etc.) relating to the user's session. The Session-Id AVP is a means for the client and servers to correlate a Diameter message with a user session.

When a Diameter server authorizes a user to implement network resources for a finite amount of time, and it is willing to extend the authorization via a future request, it MUST add the Authorization-Lifetime AVP to the answer message. The Authorization-Lifetime AVP defines the maximum number of seconds a user MAY make use of the resources before another authorization request is expected by the server. The Auth-Grace-Period AVP contains the number of seconds following the expiration of the Authorization-Lifetime, after which the server will release all state information related to the user's session. Note that if payment for services is expected by the serving realm from the user's home realm, the Authorization-Lifetime AVP, combined with the Auth-Grace-Period AVP, implies the maximum length of the session for which the home realm is willing to be fiscally responsible. Services provided past the expiration of the Authorization-Lifetime and Auth-Grace-Period AVPs are the responsibility of the access device. Of course, the actual cost of services rendered is clearly outside the scope of the protocol.

An access device that does not expect to send a re-authorization or a session termination request to the server MAY include the Auth-Session-State AVP with the value set to NO_STATE_MAINTAINED as a hint to the server. If the server accepts the hint, it agrees that since no session termination message will be received once service to the user is terminated, it cannot maintain state for the session. If the answer message from the server contains a different value in the Auth-Session-State AVP (or the default value if the AVP is absent), the access device MUST follow the server's directives. Note that the value NO_STATE_MAINTAINED MUST NOT be set in subsequent re-authorization requests and answers.

Fajardo, et al.

Standards Track

[Page 98]

RFC 6733

Diameter Base Protocol

October 2012

The base protocol does not include any authorization request messages, since these are largely application-specific and are defined in a Diameter application document. However, the base protocol does define a set of messages that are used to terminate user sessions. These are used to allow servers that maintain state information to free resources.

When a service only makes use of the accounting portion of the Diameter protocol, even in combination with an application, the Session-Id is still used to identify user sessions. However, the session termination messages are not used, since a session is signaled as being terminated by issuing an accounting stop message.

Diameter may also be used for services that cannot be easily categorized as authentication, authorization, or accounting (e.g., certain Third Generation Partnership Project Internet Multimedia System (3GPP IMS) interfaces). In such cases, the finite state machine defined in subsequent sections may not be applicable. Therefore, the application itself MAY need to define its own finite state machine. However, such application-specific state machines

1.4 Standards Compliance

SHOULD follow the general state machine framework outlined in this document such as the use of Session-Id AVPs and the use of STR/STA, ASR/ASA messages for stateful sessions.

8.1. Authorization Session State Machine

This section contains a set of finite state machines, which represent the life cycle of Diameter sessions and which MUST be observed by all Diameter implementations that make use of the authentication and/or authorization portion of a Diameter application. The term "Service-Specific" below refers to a message defined in a Diameter application (e.g., Mobile IPv4, NASREQ).

There are four different authorization session state machines supported in the Diameter base protocol. The first two describe a session in which the server is maintaining session state, indicated by the value of the Auth-Session-State AVP (or its absence). One describes the session from a client perspective, the other from a server perspective. The second two state machines are used when the server does not maintain session state. Here again, one describes the session from a client perspective, the other from a server perspective.

When a session is moved to the Idle state, any resources that were allocated for the particular session must be released. Any event not listed in the state machines MUST be considered an error condition, and an answer, if applicable, MUST be returned to the originator of the message.

Fajardo, et al.

Standards Track

[Page 99]

RFC 6733

Diameter Base Protocol

October 2012

In the case that an application does not support re-auth, the state transitions related to server-initiated re-auth, when both client and server sessions maintain state (e.g., Send RAR, Pending, Receive RAA), MAY be ignored.

In the state table, the event "Failure to send X" means that the Diameter agent is unable to send command X to the desired destination. This could be due to the peer being down or due to the peer sending back a transient failure or temporary protocol error notification DIAMETER_TOO_BUSY or DIAMETER_LOOP_DETECTED in the Result-Code AVP of the corresponding Answer command. The event 'X successfully sent' is the complement of 'Failure to send X'.

The following state machine is observed by a client when state is maintained on the server:

CLIENT, STATEFUL			
State	Event	Action	New State
Idle	Client or device requests access	Send service-specific auth req	Pending
Idle	ASR Received for unknown session	Send ASA with Result-Code = UNKNOWN_SESSION_ID	Idle

Idle	RAR Received for unknown session	Send RAA with Result-Code = UNKNOWN_SESSION_ID	Idle
Pending	Successful service-specific authorization answer received with default Auth-Session-State value	Grant Access	Open
Pending	Successful service-specific authorization answer received, but service not provided	Sent STR	Discon
Pending	Error processing successful service-specific authorization answer	Sent STR	Discon
Fajardo, et al. Standards Track [Page 100]			
RFC 6733 Diameter Base Protocol October 2012			
Pending	Failed service-specific authorization answer received	Clean up	Idle
Open	User or client device requests access to service	Send service-specific auth req	Open
Open	Successful service-specific authorization answer received	Provide service	Open
Open	Failed service-specific authorization answer received.	Discon. user/device	Idle
Open	RAR received and client will perform subsequent re-auth	Send RAA with Result-Code = SUCCESS	Open
Open	RAR received and client will not perform subsequent re-auth	Send RAA with Result-Code != SUCCESS, Discon. user/device	Idle
Open	Session-Timeout expires on access device	Send STR	Discon
Open	ASR received, client will comply with request to end the session	Send ASA with Result-Code = SUCCESS, Send STR.	Discon
Open	ASR Received, client will not comply	Send ASA with	Open

1.4 Standards Compliance

	with request to end the session	Result-Code != SUCCESS	
Open	Authorization-Lifetime + Auth-Grace-Period expires on access device	Send STR	Discon
Discon	ASR received	Send ASA	Discon

Fajardo, et al. Standards Track [Page 101]
 RFC 6733 Diameter Base Protocol October 2012

Discon	STA received	Discon. user/device	Idle
--------	--------------	------------------------	------

The following state machine is observed by a server when it is maintaining state for the session:

SERVER, STATEFUL				
State	Event	Action	New State	
Idle	Service-specific authorization request received, and user is authorized	Send successful service-specific answer	Open	
Idle	Service-specific authorization request received, and user is not authorized	Send failed service-specific answer	Idle	
Open	Service-specific authorization request received, and user is authorized	Send successful service-specific answer	Open	
Open	Service-specific authorization request received, and user is not authorized	Send failed service-specific answer, Clean up	Idle	
Open	Home server wants to confirm authentication and/or authorization of the user	Send RAR	Pending	
Pending	Received RAA with a failed Result-Code	Clean up	Idle	
Pending	Received RAA with Result-Code = SUCCESS	Update session	Open	
Open	Home server wants to terminate the service	Send ASR	Discon	

Fajardo, et al. Standards Track [Page 102]
 RFC 6733 Diameter Base Protocol October 2012

Open	Authorization-Lifetime (and Auth-Grace-Period) expires on home server	Clean up	Idle
Open	Session-Timeout expires on home server	Clean up	Idle
Discon	Failure to send ASR	Wait, resend ASR	Discon
Discon	ASR successfully sent and ASA Received with Result-Code	Clean up	Idle
Not Discon	ASA Received	None	No Change
Any	STR Received	Send STA, Clean up	Idle

The following state machine is observed by a client when state is not maintained on the server:

CLIENT, STATELESS			
State	Event	Action	New State
Idle	Client or device requests access	Send service-specific auth req	Pending
Pending	Successful service-specific authorization answer received with Auth-Session-State set to NO_STATE_MAINTAINED	Grant access	Open
Pending	Failed service-specific authorization answer received	Clean up	Idle
Open	Session-Timeout expires on access device	Discon. user/device	Idle
Open	Service to user is terminated	Discon. user/device	Idle

Fajardo, et al. Standards Track [Page 103]
 RFC 6733 Diameter Base Protocol October 2012

1.4 Standards Compliance

The following state machine is observed by a server when it is not maintaining state for the session:

SERVER, STATELESS			
State	Event	Action	New State
Idle	Service-specific authorization request received, and successfully processed	Send service-specific answer	Idle

8.2. Accounting Session State Machine

The following state machines MUST be supported for applications that have an accounting portion or that require only accounting services. The first state machine is to be observed by clients.

See Section 9.7 for Accounting Command Codes and Section 9.8 for Accounting AVPs.

The server side in the accounting state machine depends in some cases on the particular application. The Diameter base protocol defines a default state machine that MUST be followed by all applications that have not specified other state machines. This is the second state machine in this section described below.

The default server side state machine requires the reception of accounting records in any order and at any time, and it does not place any standards requirement on the processing of these records. Implementations of Diameter may perform checking, ordering, correlation, fraud detection, and other tasks based on these records. AVPs may need to be inspected as a part of these tasks. The tasks can happen either immediately after record reception or in a post-processing phase. However, as these tasks are typically application or even policy dependent, they are not standardized by the Diameter specifications. Applications MAY define requirements on when to accept accounting records based on the used value of Accounting-Realtime-Required AVP, credit-limit checks, and so on.

However, the Diameter base protocol defines one optional server side state machine that MAY be followed by applications that require keeping track of the session state at the accounting server. Note that such tracking is incompatible with the ability to sustain long duration connectivity problems. Therefore, the use of this state machine is recommended only in applications where the value of the Accounting-Realtime-Required AVP is DELIVER_AND_GRANT; hence, accounting connectivity problems are required to cause the serviced user to be disconnected. Otherwise, records produced by the client

Fajardo, et al. Standards Track [Page 104]

RFC 6733 Diameter Base Protocol October 2012

may be lost by the server, which no longer accepts them after the connectivity is re-established. This state machine is the third state machine in this section. The state machine is supervised by a supervision session timer Ts, whose value should be reasonably higher than the Acct_Interim_Interval value. Ts MAY be set to two times the value of the Acct_Interim_Interval so as to avoid the accounting session in the Diameter server to change to Idle state in case of short transient network failure.

Any event not listed in the state machines MUST be considered as an error condition, and a corresponding answer, if applicable, MUST be returned to the originator of the message.

In the state table, the event "Failure to send" means that the Diameter client is unable to communicate with the desired destination. This could be due to the peer being down, or due to the peer sending back a transient failure or temporary protocol error notification DIAMETER_OUT_OF_SPACE, DIAMETER_TOO_BUSY, or DIAMETER_LOOP_DETECTED in the Result-Code AVP of the Accounting Answer command.

The event "Failed answer" means that the Diameter client received a non-transient failure notification in the Accounting Answer command.

Note that the action "Disconnect user/dev" MUST also have an effect on the authorization session state table, e.g., cause the STR message to be sent, if the given application has both authentication/authorization and accounting portions.

The states PendingS, PendingI, PendingL, PendingE, and PendingB stand for pending states to wait for an answer to an accounting request related to a Start, Interim, Stop, Event, or buffered record, respectively.

CLIENT, ACCOUNTING				
State	Event		Action	New State
Idle	Client or device requests access		Send accounting start req.	PendingS
Idle	Client or device requests a one-time service		Send accounting event req	PendingE
Idle	Records in storage		Send record	PendingB

Fajardo, et al. Standards Track [Page 105]

RFC 6733 Diameter Base Protocol October 2012

PendingS	Successful accounting start answer received			Open
PendingS	Failure to send and buffer space available and real time not equal to DELIVER_AND_GRANT	Store Start Record		Open
PendingS	Failure to send and no buffer space available and real time equal to GRANT_AND_LOSE			Open
PendingS	Failure to send and no buffer space available and real time not equal to GRANT_AND_LOSE	Disconnect user/dev		Idle
PendingS	Failed accounting start answer received and real time equal			Open

1.4 Standards Compliance

	to GRANT_AND_LOSE		
PendingS	Failed accounting start answer received and real time not equal to GRANT_AND_LOSE	Disconnect user/dev	Idle
PendingS	User service terminated	Store stop record	PendingS
Open	Interim interval elapses	Send accounting interim record	PendingI
Open	User service terminated	Send accounting stop req.	PendingL
PendingI	Successful accounting interim answer received		Open
PendingI	Failure to send and (buffer space available or old record can be overwritten) and real time not equal to DELIVER_AND_GRANT	Store interim record	Open
Fajardo, et al. Standards Track [Page 106]			
RFC 6733 Diameter Base Protocol October 2012			
PendingI	Failure to send and no buffer space available and real time equal to GRANT_AND_LOSE		Open
PendingI	Failure to send and no buffer space available and real time not equal to GRANT_AND_LOSE	Disconnect user/dev	Idle
PendingI	Failed accounting interim answer received and real time equal to GRANT_AND_LOSE		Open
PendingI	Failed accounting interim answer received and real time not equal to GRANT_AND_LOSE	Disconnect user/dev	Idle
PendingI	User service terminated	Store stop record	PendingI
PendingE	Successful accounting event answer received		Idle
PendingE	Failure to send and buffer space available	Store event record	Idle

PendingE	Failure to send and no buffer space available		Idle
PendingE	Failed accounting event answer received		Idle
PendingB	Successful accounting answer received	Delete record	Idle
PendingB	Failure to send		Idle
PendingB	Failed accounting answer received	Delete record	Idle
PendingL	Successful accounting stop answer received		Idle
PendingL	Failure to send and buffer space available	Store stop record	Idle
Fajardo, et al. Standards Track [Page 107]			
RFC 6733 Diameter Base Protocol October 2012			
PendingL	Failure to send and no buffer space available		Idle
PendingL	Failed accounting stop answer received		Idle
SERVER, STATELESS ACCOUNTING			
State	Event	Action	New State
Idle	Accounting start request received and successfully processed.	Send accounting start answer	Idle
Idle	Accounting event request received and successfully processed.	Send accounting event answer	Idle
Idle	Interim record received and successfully processed.	Send accounting interim answer	Idle
Idle	Accounting stop request received and successfully processed	Send accounting stop answer	Idle
Idle	Accounting request received; no space left to store records	Send accounting answer; Result-Code = OUT_OF_SPACE	Idle

Fajardo, et al. Standards Track [Page 108]
 RFC 6733 Diameter Base Protocol October 2012

SERVER, STATEFUL ACCOUNTING				
State	Event	Action	New State	
Idle	Accounting start request received and successfully processed.	Send accounting start answer; Start Ts	Open	
Idle	Accounting event request received and successfully processed.	Send accounting event answer	Idle	
Idle	Accounting request received; no space left to store records	Send accounting answer; Result-Code = OUT_OF_SPACE	Idle	
Open	Interim record received and successfully processed.	Send accounting interim answer; Restart Ts	Open	
Open	Accounting stop request received and successfully processed	Send accounting stop answer; Stop Ts	Idle	
Open	Accounting request received; no space left to store records	Send accounting answer; Result-Code = OUT_OF_SPACE; Stop Ts	Idle	
Open	Session supervision timer Ts expired	Stop Ts	Idle	

1.4 Standards Compliance

A successful RAA message MUST be followed by an application-specific authentication and/or authorization message.

Message Format

```
<RAA> ::= < Diameter Header: 258, PXY >
        < Session-Id >
        { Result-Code }
        { Origin-Host }
        { Origin-Realm }
        [ User-Name ]
        [ Origin-State-Id ]
        [ Error-Message ]
        [ Error-Reporting-Host ]
        [ Failed-AVP ]
        * [ Redirect-Host ]
        [ Redirect-Host-Usage ]
        [ Redirect-Max-Cache-Time ]
        * [ Proxy-Info ]
        * [ AVP ]
```

8.4. Session Termination

It is necessary for a Diameter server that authorized a session, for which it is maintaining state, to be notified when that session is no longer active, both for tracking purposes as well as to allow stateful agents to release any resources that they may have provided for the user's session. For sessions whose state is not being maintained, this section is not used.

When a user session that required Diameter authorization terminates, the access device that provided the service MUST issue a Session-Termination-Request (STR) message to the Diameter server that authorized the service, to notify it that the session is no longer active. An STR MUST be issued when a user session terminates for any reason, including user logoff, expiration of Session-Timeout, administrative action, termination upon receipt of an Abort-Session-Request (see below), orderly shutdown of the access device, etc.

The access device also MUST issue an STR for a session that was authorized but never actually started. This could occur, for example, due to a sudden resource shortage in the access device, or because the access device is unwilling to provide the type of service requested in the authorization, or because the access device does not support a mandatory AVP returned in the authorization, etc.

It is also possible that a session that was authorized is never actually started due to action of a proxy. For example, a proxy may

modify an authorization answer, converting the result from success to failure, prior to forwarding the message to the access device. If the answer did not contain an Auth-Session-State AVP with the value NO_STATE_MAINTAINED, a proxy that causes an authorized session not to be started MUST issue an STR to the Diameter server that authorized the session, since the access device has no way of knowing that the session had been authorized.

A Diameter server that receives an STR message MUST clean up resources (e.g., session state) associated with the Session-Id specified in the STR and return a Session-Termination-Answer.

A Diameter server also MUST clean up resources when the Session-Timeout expires, or when the Authorization-Lifetime and the Auth-Grace-Period AVPs expire without receipt of a re-authorization request, regardless of whether an STR for that session is received. The access device is not expected to provide service beyond the expiration of these timers; thus, expiration of either of these timers implies that the access device may have unexpectedly shut down.

8.4.1. Session-Termination-Request

The Session-Termination-Request (STR), indicated by the Command Code set to 275 and the Command Flags' 'R' bit set, is sent by a Diameter client or by a Diameter proxy to inform the Diameter server that an authenticated and/or authorized session is being terminated.

Message Format

```
<STR> ::= < Diameter Header: 275, REQ, PXY >
         < Session-Id >
         { Origin-Host }
         { Origin-Realm }
         { Destination-Realm }
         { Auth-Application-Id }
         { Termination-Cause }
         [ User-Name ]
         [ Destination-Host ]
         * [ Class ]
         [ Origin-State-Id ]
         * [ Proxy-Info ]
         * [ Route-Record ]
         * [ AVP ]
```

8.4.2. Session-Termination-Answer

The Session-Termination-Answer (STA), indicated by the Command Code set to 275 and the message flags' 'R' bit clear, is sent by the Diameter server to acknowledge the notification that the session has been terminated. The Result-Code AVP MUST be present, and it MAY contain an indication that an error occurred while servicing the STR.

Upon sending or receipt of the STA, the Diameter server MUST release all resources for the session indicated by the Session-Id AVP. Any intermediate server in the Proxy-Chain MAY also release any resources, if necessary.

Message Format

```
<STA> ::= < Diameter Header: 275, PXY >
```

```
< Session-Id >
{ Result-Code }
{ Origin-Host }
{ Origin-Realm }
[ User-Name ]
* [ Class ]
[ Error-Message ]
[ Error-Reporting-Host ]
[ Failed-AVP ]
[ Origin-State-Id ]
* [ Redirect-Host ]
[ Redirect-Host-Usage ]
[ Redirect-Max-Cache-Time ]
* [ Proxy-Info ]
* [ AVP ]
```

8.5. Aborting a Session

A Diameter server may request that the access device stop providing service for a particular session by issuing an Abort-Session-Request (ASR).

For example, the Diameter server that originally authorized the session may be required to cause that session to be stopped for lack of credit or other reasons that were not anticipated when the session was first authorized.

An access device that receives an ASR with Session-ID equal to a currently active session MAY stop the session. Whether the access device stops the session or not is implementation and/or configuration dependent. For example, an access device may honor ASRs from certain agents only. In any case, the access device MUST

Fajardo, et al. Standards Track [Page 113]

RFC 6733 Diameter Base Protocol October 2012

respond with an Abort-Session-Answer, including a Result-Code AVP to indicate what action it took.

8.5.1. Abort-Session-Request

The Abort-Session-Request (ASR), indicated by the Command Code set to 274 and the message flags' 'R' bit set, may be sent by any Diameter server or any Diameter proxy to the access device that is providing session service, to request that the session identified by the Session-Id be stopped.

Message Format

```
<ASR> ::= < Diameter Header: 274, REQ, PXY >
< Session-Id >
{ Origin-Host }
{ Origin-Realm }
{ Destination-Realm }
{ Destination-Host }
{ Auth-Application-Id }
[ User-Name ]
[ Origin-State-Id ]
* [ Proxy-Info ]
* [ Route-Record ]
* [ AVP ]
```

8.5.2. Abort-Session-Answer

The Abort-Session-Answer (ASA), indicated by the Command Code set to 274 and the message flags' 'R' bit clear, is sent in response to the ASR. The Result-Code AVP MUST be present and indicates the disposition of the request.

If the session identified by Session-Id in the ASR was successfully terminated, the Result-Code is set to DIAMETER_SUCCESS. If the session is not currently active, the Result-Code is set to DIAMETER_UNKNOWN_SESSION_ID. If the access device does not stop the session for any other reason, the Result-Code is set to DIAMETER_UNABLE_TO_COMPLY.

Fajardo, et al.

Standards Track

[Page 114]

RFC 6733

Diameter Base Protocol

October 2012

Message Format

```
<ASA> ::= < Diameter Header: 274, PXY >
         < Session-Id >
         { Result-Code }
         { Origin-Host }
         { Origin-Realm }
         [ User-Name ]
         [ Origin-State-Id ]
         [ Error-Message ]
         [ Error-Reporting-Host ]
         [ Failed-AVP ]
         * [ Redirect-Host ]
         [ Redirect-Host-Usage ]
         [ Redirect-Max-Cache-Time ]
         * [ Proxy-Info ]
         * [ AVP ]
```

8.6. Inferring Session Termination from Origin-State-Id

The Origin-State-Id is used to allow detection of terminated sessions for which no STR would have been issued, due to unanticipated shutdown of an access device.

A Diameter client or access device increments the value of the Origin-State-Id every time it is started or powered up. The new Origin-State-Id is then sent in the CER/CEA message immediately upon connection to the server. The Diameter server receiving the new Origin-State-Id can determine whether the sending Diameter client had abruptly shut down by comparing the old value of the Origin-State-Id it has kept for that specific client is less than the new value and whether it has un-terminated sessions originating from that client.

An access device can also include the Origin-State-Id in request

messages other than the CER if there are relays or proxies in between the access device and the server. In this case, however, the server cannot discover that the access device has been restarted unless and until it receives a new request from it. Therefore, this mechanism is more opportunistic across proxies and relays.

The Diameter server may assume that all sessions that were active prior to detection of a client restart have been terminated. The Diameter server MAY clean up all session state associated with such lost sessions, and it MAY also issue STRs for all such lost sessions that were authorized on upstream servers, to allow session state to be cleaned up globally.

Fajardo, et al. Standards Track [Page 115]
RFC 6733 Diameter Base Protocol October 2012

8.7. Auth-Request-Type AVP

The Auth-Request-Type AVP (AVP Code 274) is of type Enumerated and is included in application-specific auth requests to inform the peers whether a user is to be authenticated only, authorized only, or both. Note any value other than both MAY cause RADIUS interoperability issues. The following values are defined:

AUTHENTICATE_ONLY 1

The request being sent is for authentication only, and it MUST contain the relevant application-specific authentication AVPs that are needed by the Diameter server to authenticate the user.

AUTHORIZE_ONLY 2

The request being sent is for authorization only, and it MUST contain the application-specific authorization AVPs that are necessary to identify the service being requested/offered.

AUTHORIZE_AUTHENTICATE 3

The request contains a request for both authentication and authorization. The request MUST include both the relevant application-specific authentication information and authorization information necessary to identify the service being requested/offered.

8.8. Session-Id AVP

The Session-Id AVP (AVP Code 263) is of type UTF8String and is used to identify a specific session (see Section 8). All messages pertaining to a specific session MUST include only one Session-Id AVP, and the same value MUST be used throughout the life of a session. When present, the Session-Id SHOULD appear immediately following the Diameter header (see Section 3).

The Session-Id MUST be globally and eternally unique, as it is meant to uniquely identify a user session without reference to any other information, and it may be needed to correlate historical authentication information with accounting information. The Session-Id includes a mandatory portion and an implementation-defined portion; a recommended format for the implementation-defined portion

1.4 Standards Compliance

RFC 6733

Diameter Base Protocol

October 2012

If both this AVP and the Session-Timeout AVP are present in a message, the value of the latter MUST NOT be smaller than the Authorization-Lifetime AVP.

An Authorization-Lifetime AVP MAY be present in re-authorization messages, and it contains the number of seconds the user is authorized to receive service from the time the re-auth answer message is received by the access device.

This AVP MAY be provided by the client as a hint of the maximum lifetime that it is willing to accept. The server MUST return a value that is equal to, or smaller than, the one provided by the client.

8.10. Auth-Grace-Period AVP

The Auth-Grace-Period AVP (AVP Code 276) is of type Unsigned32 and contains the number of seconds the Diameter server will wait following the expiration of the Authorization-Lifetime AVP before cleaning up resources for the session.

8.11. Auth-Session-State AVP

The Auth-Session-State AVP (AVP Code 277) is of type Enumerated and specifies whether state is maintained for a particular session. The client MAY include this AVP in requests as a hint to the server, but the value in the server's answer message is binding. The following values are supported:

STATE_MAINTAINED 0

This value is used to specify that session state is being maintained, and the access device MUST issue a session termination message when service to the user is terminated. This is the default value.

NO_STATE_MAINTAINED 1

This value is used to specify that no session termination messages will be sent by the access device upon expiration of the Authorization-Lifetime.

8.12. Re-Auth-Request-Type AVP

The Re-Auth-Request-Type AVP (AVP Code 285) is of type Enumerated and is included in application-specific auth answers to inform the client of the action expected upon expiration of the Authorization-Lifetime.

Fajardo, et al.

Standards Track

[Page 118]

RFC 6733

Diameter Base Protocol

October 2012

If the answer message contains an Authorization-Lifetime AVP with a positive value, the Re-Auth-Request-Type AVP MUST be present in an answer message. The following values are defined:

AUTHORIZE_ONLY 0

An authorization only re-auth is expected upon expiration of the Authorization-Lifetime. This is the default value if the AVP is not present in answer messages that include the Authorization-Lifetime.

AUTHORIZE_AUTHENTICATE 1

An authentication and authorization re-auth is expected upon expiration of the Authorization-Lifetime.

8.13. Session-Timeout AVP

The Session-Timeout AVP (AVP Code 27) [RFC2865] is of type Unsigned32 and contains the maximum number of seconds of service to be provided to the user before termination of the session. When both the Session-Timeout and the Authorization-Lifetime AVPs are present in an answer message, the former MUST be equal to or greater than the value of the latter.

A session that terminates on an access device due to the expiration of the Session-Timeout MUST cause an STR to be issued, unless both the access device and the home server had previously agreed that no session termination messages would be sent (see Section 8).

A Session-Timeout AVP MAY be present in a re-authorization answer message, and it contains the remaining number of seconds from the beginning of the re-auth.

A value of zero, or the absence of this AVP, means that this session has an unlimited number of seconds before termination.

This AVP MAY be provided by the client as a hint of the maximum timeout that it is willing to accept. However, the server MAY return a value that is equal to, or smaller than, the one provided by the client.

8.14. User-Name AVP

The User-Name AVP (AVP Code 1) [RFC2865] is of type UTF8String, which contains the User-Name, in a format consistent with the NAI specification [RFC4282].

Fajardo, et al. Standards Track [Page 119]

RFC 6733 Diameter Base Protocol October 2012

8.15. Termination-Cause AVP

The Termination-Cause AVP (AVP Code 295) is of type Enumerated, and is used to indicate the reason why a session was terminated on the access device. The currently assigned values for this AVP can be found in the IANA registry for Termination-Cause AVP Values [IANATCV].

8.16. Origin-State-Id AVP

The Origin-State-Id AVP (AVP Code 278), of type Unsigned32, is a monotonically increasing value that is advanced whenever a Diameter entity restarts with loss of previous state, for example, upon reboot. Origin-State-Id MAY be included in any Diameter message,

1.4 Standards Compliance

including CER.

A Diameter entity issuing this AVP MUST create a higher value for this AVP each time its state is reset. A Diameter entity MAY set Origin-State-Id to the time of startup, or it MAY use an incrementing counter retained in non-volatile memory across restarts.

The Origin-State-Id, if present, MUST reflect the state of the entity indicated by Origin-Host. If a proxy modifies Origin-Host, it MUST either remove Origin-State-Id or modify it appropriately as well. Typically, Origin-State-Id is used by an access device that always starts up with no active sessions; that is, any session active prior to restart will have been lost. By including Origin-State-Id in a message, it allows other Diameter entities to infer that sessions associated with a lower Origin-State-Id are no longer active. If an access device does not intend for such inferences to be made, it MUST either not include Origin-State-Id in any message or set its value to 0.

8.17. Session-Binding AVP

The Session-Binding AVP (AVP Code 270) is of type Unsigned32, and it MAY be present in application-specific authorization answer messages. If present, this AVP MAY inform the Diameter client that all future application-specific re-auth and Session-Termination-Request messages for this session MUST be sent to the same authorization server.

Fajardo, et al.

Standards Track

[Page 120]

RFC 6733

Diameter Base Protocol

October 2012

This field is a bit mask, and the following bits have been defined:

RE_AUTH 1

When set, future re-auth messages for this session MUST NOT include the Destination-Host AVP. When cleared, the default value, the Destination-Host AVP MUST be present in all re-auth messages for this session.

STR 2

When set, the STR message for this session MUST NOT include the Destination-Host AVP. When cleared, the default value, the Destination-Host AVP MUST be present in the STR message for this session.

ACCOUNTING 4

When set, all accounting messages for this session MUST NOT include the Destination-Host AVP. When cleared, the default value, the Destination-Host AVP, if known, MUST be present in all accounting messages for this session.

8.18. Session-Server-Failover AVP

The Session-Server-Failover AVP (AVP Code 271) is of type Enumerated and MAY be present in application-specific authorization answer messages that either do not include the Session-Binding AVP or include the Session-Binding AVP with any of the bits set to a zero value. If present, this AVP MAY inform the Diameter client that if a re-auth or STR message fails due to a delivery problem, the Diameter client SHOULD issue a subsequent message without the Destination-Host AVP. When absent, the default value is REFUSE_SERVICE.

The following values are supported:

REFUSE_SERVICE 0

If either the re-auth or the STR message delivery fails, terminate service with the user and do not attempt any subsequent attempts.

Fajardo, et al.

Standards Track

[Page 121]

RFC 6733

Diameter Base Protocol

October 2012

TRY_AGAIN 1

If either the re-auth or the STR message delivery fails, resend the failed message without the Destination-Host AVP present.

ALLOW_SERVICE 2

If re-auth message delivery fails, assume that re-authorization succeeded. If STR message delivery fails, terminate the session.

TRY_AGAIN_ALLOW_SERVICE 3

If either the re-auth or the STR message delivery fails, resend the failed message without the Destination-Host AVP present. If the second delivery fails for re-auth, assume re-authorization succeeded. If the second delivery fails for STR, terminate the session.

8.19. Multi-Round-Time-Out AVP

The Multi-Round-Time-Out AVP (AVP Code 272) is of type Unsigned32 and SHOULD be present in application-specific authorization answer messages whose Result-Code AVP is set to DIAMETER_MULTI_ROUND_AUTH. This AVP contains the maximum number of seconds that the access device MUST provide the user in responding to an authentication request.

8.20. Class AVP

The Class AVP (AVP Code 25) is of type OctetString and is used by Diameter servers to return state information to the access device. When one or more Class AVPs are present in application-specific

1.4 Standards Compliance

authorization answer messages, they MUST be present in subsequent re-authorization, session termination and accounting messages. Class AVPs found in a re-authorization answer message override the ones found in any previous authorization answer message. Diameter server implementations SHOULD NOT return Class AVPs that require more than 4096 bytes of storage on the Diameter client. A Diameter client that receives Class AVPs whose size exceeds local available storage MUST terminate the session.

8.21. Event-Timestamp AVP

The Event-Timestamp (AVP Code 55) is of type Time and MAY be included in an Accounting-Request and Accounting-Answer messages to record the time that the reported event occurred, in seconds since January 1, 1900 00:00 UTC.

Fajardo, et al. Standards Track [Page 122]

RFC 6733 Diameter Base Protocol October 2012

9. Accounting

This accounting protocol is based on a server directed model with capabilities for real-time delivery of accounting information. Several fault resilience methods [RFC2975] have been built into the protocol in order minimize loss of accounting data in various fault situations and under different assumptions about the capabilities of the used devices.

9.1. Server Directed Model

The server directed model means that the device generating the accounting data gets information from either the authorization server (if contacted) or the accounting server regarding the way accounting data shall be forwarded. This information includes accounting record timeliness requirements.

As discussed in [RFC2975], real-time transfer of accounting records is a requirement, such as the need to perform credit-limit checks and fraud detection. Note that batch accounting is not a requirement, and is therefore not supported by Diameter. Should batched accounting be required in the future, a new Diameter application will need to be created, or it could be handled using another protocol. Note, however, that even if at the Diameter layer, accounting requests are processed one by one; transport protocols used under Diameter typically batch several requests in the same packet under heavy traffic conditions. This may be sufficient for many applications.

The authorization server (chain) directs the selection of proper transfer strategy, based on its knowledge of the user and relationships of roaming partnerships. The server (or agents) uses the Acct-Interim-Interval and Accounting-Realtime-Required AVPs to control the operation of the Diameter peer operating as a client. The Acct-Interim-Interval AVP, when present, instructs the Diameter node acting as a client to produce accounting records continuously even during a session. Accounting-Realtime-Required AVP is used to control the behavior of the client when the transfer of accounting records from the Diameter client is delayed or unsuccessful.

The Diameter accounting server MAY override the interim interval or

1.4 Standards Compliance

Fajardo, et al. Standards Track [Page 124]

RFC 6733 Diameter Base Protocol October 2012

9.4. Fault Resilience

Diameter base protocol mechanisms are used to overcome small message loss and network faults of a temporary nature.

Diameter peers acting as clients MUST implement the use of failover to guard against server failures and certain network failures. Diameter peers acting as agents or related off-line processing systems MUST detect duplicate accounting records caused by the sending of the same record to several servers and duplication of messages in transit. This detection MUST be based on the inspection of the Session-Id and Accounting-Record-Number AVP pairs. Appendix C discusses duplicate detection needs and implementation issues.

Diameter clients MAY have non-volatile memory for the safe storage of accounting records over reboots or extended network failures, network partitions, and server failures. If such memory is available, the client SHOULD store new accounting records there as soon as the records are created and until a positive acknowledgement of their reception from the Diameter server has been received. Upon a reboot, the client MUST start sending the records in the non-volatile memory to the accounting server with the appropriate modifications in termination cause, session length, and other relevant information in the records.

A further application of this protocol may include AVPs to control the maximum number of accounting records that may be stored in the Diameter client without committing them to the non-volatile memory or transferring them to the Diameter server.

The client SHOULD NOT remove the accounting data from any of its memory areas before the correct Accounting-Answer has been received. The client MAY remove the oldest, undelivered, or as yet unacknowledged accounting data if it runs out of resources such as memory. It is an implementation-dependent matter for the client to accept new sessions under this condition.

9.5. Accounting Records

In all accounting records, the Session-Id AVP MUST be present; the User-Name AVP MUST be present if it is available to the Diameter client.

Different types of accounting records are sent depending on the actual type of accounted service and the authorization server's directions for interim accounting. If the accounted service is a

Fajardo, et al. Standards Track [Page 125]

RFC 6733 Diameter Base Protocol October 2012

one-time event, meaning that the start and stop of the event are simultaneous, then the Accounting-Record-Type AVP MUST be present and set to the value EVENT_RECORD.

If the accounted service is of a measurable length, then the AVP MUST use the values START_RECORD, STOP_RECORD, and possibly, INTERIM_RECORD. If the authorization server has not directed interim accounting to be enabled for the session, two accounting records MUST be generated for each service of type session. When the initial Accounting-Request for a given session is sent, the Accounting-Record-Type AVP MUST be set to the value START_RECORD. When the last Accounting-Request is sent, the value MUST be STOP_RECORD.

If the authorization server has directed interim accounting to be enabled, the Diameter client MUST produce additional records between the START_RECORD and STOP_RECORD, marked INTERIM_RECORD. The production of these records is directed by Acct-Interim-Interval as well as any re-authentication or re-authorization of the session. The Diameter client MUST overwrite any previous interim accounting records that are locally stored for delivery, if a new record is being generated for the same session. This ensures that only one pending interim record can exist on an access device for any given session.

A particular value of Accounting-Sub-Session-Id MUST appear only in one sequence of accounting records from a Diameter client, except for the purposes of retransmission. The one sequence that is sent MUST be either one record with Accounting-Record-Type AVP set to the value EVENT_RECORD or several records starting with one having the value START_RECORD, followed by zero or more INTERIM_RECORDs and a single STOP_RECORD. A particular Diameter application specification MUST define the type of sequences that MUST be used.

9.6. Correlation of Accounting Records

If an application uses accounting messages, it can correlate accounting records with a specific application session by using the Session-Id of the particular application session in the accounting messages. Accounting messages MAY also use a different Session-Id from that of the application sessions, in which case, other session-related information is needed to perform correlation.

In cases where an application requires multiple accounting sub-sessions, an Accounting-Sub-Session-Id AVP is used to differentiate each sub-session. The Session-Id would remain constant for all sub-sessions and is used to correlate all the sub-sessions to a particular application session. Note that receiving a STOP_RECORD

Fajardo, et al.

Standards Track

[Page 126]

RFC 6733

Diameter Base Protocol

October 2012

with no Accounting-Sub-Session-Id AVP when sub-sessions were originally used in the START_RECORD messages implies that all sub-sessions are terminated.

There are also cases where an application needs to correlate multiple application sessions into a single accounting record; the accounting record may span multiple different Diameter applications and sessions used by the same user at a given time. In such cases, the Acct-Multi-Session-Id AVP is used. The Acct-Multi-Session-Id AVP SHOULD be signaled by the server to the access device (typically, during authorization) when it determines that a request belongs to an existing session. The access device MUST then include the Acct-

1.4 Standards Compliance

Multi-Session-Id AVP in all subsequent accounting messages.

The Acct-Multi-Session-Id AVP MAY include the value of the original Session-Id. Its contents are implementation specific, but the MUST be globally unique across other Acct-Multi-Session-Ids and MUST NOT change during the life of a session.

A Diameter application document MUST define the exact concept of a session that is being accounted, and it MAY define the concept of a multi-session. For instance, the NASREQ DIAMETER application treats a single PPP connection to a Network Access Server as one session and a set of Multilink PPP sessions as one multi-session.

9.7. Accounting Command Codes

This section defines Command Code values that MUST be supported by all Diameter implementations that provide accounting services.

9.7.1. Accounting-Request

The Accounting-Request (ACR) command, indicated by the Command Code field set to 271 and the Command Flags' 'R' bit set, is sent by a Diameter node, acting as a client, in order to exchange accounting information with a peer.

In addition to the AVPs listed below, Accounting-Request messages SHOULD include service-specific accounting AVPs.

Fajardo, et al.

Standards Track

[Page 127]

RFC 6733

Diameter Base Protocol

October 2012

Message Format

```
<ACR> ::= < Diameter Header: 271, REQ, PXY >
  < Session-Id >
  { Origin-Host }
  { Origin-Realm }
  { Destination-Realm }
  { Accounting-Record-Type }
  { Accounting-Record-Number }
  [ Acct-Application-Id ]
  [ Vendor-Specific-Application-Id ]
  [ User-Name ]
  [ Destination-Host ]
  [ Accounting-Sub-Session-Id ]
  [ Acct-Session-Id ]
  [ Acct-Multi-Session-Id ]
  [ Acct-Interim-Interval ]
  [ Accounting-Realtime-Required ]
  [ Origin-State-Id ]
  [ Event-Timestamp ]
  * [ Proxy-Info ]
```

```
* [ Route-Record ]
* [ AVP ]
```

9.7.2. Accounting-Answer

The Accounting-Answer (ACA) command, indicated by the Command Code field set to 271 and the Command Flags' 'R' bit cleared, is used to acknowledge an Accounting-Request command. The Accounting-Answer command contains the same Session-Id as the corresponding request.

Only the target Diameter server, known as the home Diameter server, SHOULD respond with the Accounting-Answer command.

In addition to the AVPs listed below, Accounting-Answer messages SHOULD include service-specific accounting AVPs.

Fajardo, et al.

Standards Track

[Page 128]

RFC 6733

Diameter Base Protocol

October 2012

Message Format

```
<ACA> ::= < Diameter Header: 271, PXY >
  < Session-Id >
  { Result-Code }
  { Origin-Host }
  { Origin-Realm }
  { Accounting-Record-Type }
  { Accounting-Record-Number }
  [ Acct-Application-Id ]
  [ Vendor-Specific-Application-Id ]
  [ User-Name ]
  [ Accounting-Sub-Session-Id ]
  [ Acct-Session-Id ]
  [ Acct-Multi-Session-Id ]
  [ Error-Message ]
  [ Error-Reporting-Host ]
  [ Failed-AVP ]
  [ Acct-Interim-Interval ]
  [ Accounting-Realtime-Required ]
  [ Origin-State-Id ]
  [ Event-Timestamp ]
  * [ Proxy-Info ]
  * [ AVP ]
```

9.8. Accounting AVPs

This section contains AVPs that describe accounting usage information related to a specific session.

9.8.1. Accounting-Record-Type AVP

The Accounting-Record-Type AVP (AVP Code 480) is of type Enumerated and contains the type of accounting record being sent. The following values are currently defined for the Accounting-Record-Type AVP:

EVENT_RECORD 1

An Accounting Event Record is used to indicate that a one-time event has occurred (meaning that the start and end of the event are simultaneous). This record contains all information relevant to the service, and it is the only record of the service.

Fajardo, et al.

Standards Track

[Page 129]

RFC 6733

Diameter Base Protocol

October 2012

START_RECORD 2

Accounting Start, Interim, and Stop Records are used to indicate that a service of a measurable length has been given. An Accounting Start Record is used to initiate an accounting session and contains accounting information that is relevant to the initiation of the session.

INTERIM_RECORD 3

An Interim Accounting Record contains cumulative accounting information for an existing accounting session. Interim Accounting Records SHOULD be sent every time a re-authentication or re-authorization occurs. Further, additional interim record triggers MAY be defined by application-specific Diameter applications. The selection of whether to use INTERIM_RECORD records is done by the Acct-Interim-Interval AVP.

STOP_RECORD 4

An Accounting Stop Record is sent to terminate an accounting session and contains cumulative accounting information relevant to the existing session.

9.8.2. Acct-Interim-Interval AVP

The Acct-Interim-Interval AVP (AVP Code 85) is of type Unsigned32 and is sent from the Diameter home authorization server to the Diameter client. The client uses information in this AVP to decide how and when to produce accounting records. With different values in this AVP, service sessions can result in one, two, or two+N accounting records, based on the needs of the home organization. The following accounting record production behavior is directed by the inclusion of this AVP:

1. The omission of the Acct-Interim-Interval AVP or its inclusion with Value field set to 0 means that EVENT_RECORD, START_RECORD, and STOP_RECORD are produced, as appropriate for the service.

2. The inclusion of the AVP with Value field set to a non-zero value means that INTERIM_RECORD records MUST be produced between the START_RECORD and STOP_RECORD records. The Value field of this AVP is the nominal interval between these records in seconds. The Diameter node that originates the accounting information, known as the client, MUST produce the first INTERIM_RECORD record roughly at the time when this nominal interval has elapsed from

Fajardo, et al. Standards Track [Page 130]

RFC 6733 Diameter Base Protocol October 2012

the START_RECORD, the next one again as the interval has elapsed once more, and so on until the session ends and a STOP_RECORD record is produced.

The client MUST ensure that the interim record production times are randomized so that large accounting message storms are not created either among records or around a common service start time.

9.8.3. Accounting-Record-Number AVP

The Accounting-Record-Number AVP (AVP Code 485) is of type Unsigned32 and identifies this record within one session. As Session-Id AVPs are globally unique, the combination of Session-Id and Accounting-Record-Number AVPs is also globally unique and can be used in matching accounting records with confirmations. An easy way to produce unique numbers is to set the value to 0 for records of type EVENT_RECORD and START_RECORD and set the value to 1 for the first INTERIM_RECORD, 2 for the second, and so on until the value for STOP_RECORD is one more than for the last INTERIM_RECORD.

9.8.4. Acct-Session-Id AVP

The Acct-Session-Id AVP (AVP Code 44) is of type OctetString is only used when RADIUS/Diameter translation occurs. This AVP contains the contents of the RADIUS Acct-Session-Id attribute.

9.8.5. Acct-Multi-Session-Id AVP

The Acct-Multi-Session-Id AVP (AVP Code 50) is of type UTF8String, following the format specified in Section 8.8. The Acct-Multi-Session-Id AVP is used to link multiple related accounting sessions, where each session would have a unique Session-Id but the same Acct-Multi-Session-Id AVP. This AVP MAY be returned by the Diameter server in an authorization answer, and it MUST be used in all accounting messages for the given session.

9.8.6. Accounting-Sub-Session-Id AVP

The Accounting-Sub-Session-Id AVP (AVP Code 287) is of type Unsigned64 and contains the accounting sub-session identifier. The combination of the Session-Id and this AVP MUST be unique per sub-session, and the value of this AVP MUST be monotonically increased by one for all new sub-sessions. The absence of this AVP implies no sub-sessions are in use, with the exception of an Accounting-Request whose Accounting-Record-Type is set to STOP_RECORD. A STOP_RECORD message with no Accounting-Sub-Session-Id AVP present will signal the termination of all sub-sessions for a given Session-Id.

Fajardo, et al. Standards Track [Page 131]

RFC 6733 Diameter Base Protocol October 2012

9.8.7. Accounting-Realtime-Required AVP

The Accounting-Realtime-Required AVP (AVP Code 483) is of type Enumerated and is sent from the Diameter home authorization server to the Diameter client or in the Accounting-Answer from the accounting server. The client uses information in this AVP to decide what to do if the sending of accounting records to the accounting server has been temporarily prevented due to, for instance, a network problem.

DELIVER_AND_GRANT 1

The AVP with Value field set to DELIVER_AND_GRANT means that the service MUST only be granted as long as there is a connection to an accounting server. Note that the set of alternative accounting servers are treated as one server in this sense. Having to move the accounting record stream to a backup server is not a reason to discontinue the service to the user.

GRANT_AND_STORE 2

The AVP with Value field set to GRANT_AND_STORE means that service SHOULD be granted if there is a connection, or as long as records can still be stored as described in Section 9.4.

This is the default behavior if the AVP isn't included in the reply from the authorization server.

GRANT_AND_LOSE 3

The AVP with Value field set to GRANT_AND_LOSE means that service SHOULD be granted even if the records cannot be delivered or stored.

10. AVP Occurrence Tables

The following tables present the AVPs defined in this document and specify in which Diameter messages they MAY or MAY NOT be present. AVPs that occur only inside a Grouped AVP are not shown in these tables.

The tables use the following symbols:

- 0 The AVP MUST NOT be present in the message.
- 0+ Zero or more instances of the AVP MAY be present in the message.

Fajardo, et al. Standards Track [Page 132]

RFC 6733 Diameter Base Protocol October 2012

0-1 Zero or one instance of the AVP MAY be present in the message.

It is considered an error if there are more than one instance of the AVP.

- 1 One instance of the AVP MUST be present in the message.
- 1+ At least one instance of the AVP MUST be present in the message.

10.1. Base Protocol Command AVP Table

The table in this section is limited to the non-Accounting Command Codes defined in this specification.

Attribute Name	Command Code											
	CER	CEA	DPR	DPA	DWR	DWA	RAR	RAA	ASR	ASA	STR	STA
Acct-Interim-Interval	0	0	0	0	0	0	0-1	0	0	0	0	0
Accounting-Realtime-Required	0	0	0	0	0	0	0-1	0	0	0	0	0
Acct-Application-Id	0+	0+	0	0	0	0	0	0	0	0	0	0
Auth-Application-Id	0+	0+	0	0	0	0	1	0	1	0	1	0
Auth-Grace-Period	0	0	0	0	0	0	0	0	0	0	0	0
Auth-Request-Type	0	0	0	0	0	0	0	0	0	0	0	0
Auth-Session-State	0	0	0	0	0	0	0	0	0	0	0	0
Authorization-Lifetime	0	0	0	0	0	0	0	0	0	0	0	0
Class	0	0	0	0	0	0	0	0	0	0	0+	0+
Destination-Host	0	0	0	0	0	0	1	0	1	0	0-1	0
Destination-Realm	0	0	0	0	0	0	1	0	1	0	1	0
Disconnect-Cause	0	0	1	0	0	0	0	0	0	0	0	0
Error-Message	0	0-1	0	0-1	0	0-1	0	0-1	0	0-1	0	0-1
Error-Reporting-Host	0	0	0	0	0	0	0	0-1	0	0-1	0	0-1
Failed-AVP	0	0-1	0	0-1	0	0-1	0	0-1	0	0-1	0	0-1
Firmware-Revision	0-1	0-1	0	0	0	0	0	0	0	0	0	0
Host-IP-Address	1+	1+	0	0	0	0	0	0	0	0	0	0
Inband-Security-Id	0	0	0	0	0	0	0	0	0	0	0	0
Multi-Round-Time-Out	0	0	0	0	0	0	0	0	0	0	0	0

Fajardo, et al.

Standards Track

[Page 133]

RFC 6733

Diameter Base Protocol

October 2012

Origin-Host	1	1	1	1	1	1	1	1	1	1	1	1
Origin-Realm	1	1	1	1	1	1	1	1	1	1	1	1
Origin-State-Id	0-1	0-1	0	0	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1
Product-Name	1	1	0	0	0	0	0	0	0	0	0	0
Proxy-Info	0	0	0	0	0	0	0+	0+	0+	0+	0+	0+
Redirect-Host	0	0	0	0	0	0	0	0+	0	0+	0	0+
Redirect-Host-Usage	0	0	0	0	0	0	0	0-1	0	0-1	0	0-1
Redirect-Max-Cache-Time	0	0	0	0	0	0	0	0-1	0	0-1	0	0-1
Result-Code	0	1	0	1	0	1	0	1	0	1	0	1

1.4 Standards Compliance

Re-Auth-Request-Type	0	0	0	0	0	0	1	0	0	0	0	0
Route-Record	0	0	0	0	0	0	0+	0	0+	0	0+	0
Session-Binding	0	0	0	0	0	0	0	0	0	0	0	0
Session-Id	0	0	0	0	0	0	1	1	1	1	1	1
Session-Server-Failover	0	0	0	0	0	0	0	0	0	0	0	0
Session-Timeout	0	0	0	0	0	0	0	0	0	0	0	0
Supported-Vendor-Id	0+	0+	0	0	0	0	0	0	0	0	0	0
Termination-Cause	0	0	0	0	0	0	0	0	0	0	1	0
User-Name	0	0	0	0	0	0	0-1	0-1	0-1	0-1	0-1	0-1
Vendor-Id	1	1	0	0	0	0	0	0	0	0	0	0
Vendor-Specific-Application-Id	0+	0+	0	0	0	0	0	0	0	0	0	0

10.2. Accounting AVP Table

The table in this section is used to represent which AVPs defined in this document are to be present in the Accounting messages. These AVP occurrence requirements are guidelines, which may be expanded, and/or overridden by application-specific requirements in the Diameter applications documents.

Attribute Name	Command Code	
	ACR	ACA
Acct-Interim-Interval	0-1	0-1
Acct-Multi-Session-Id	0-1	0-1
Accounting-Record-Number	1	1
Accounting-Record-Type	1	1
Acct-Session-Id	0-1	0-1
Accounting-Sub-Session-Id	0-1	0-1
Accounting-Realtime-Required	0-1	0-1
Acct-Application-Id	0-1	0-1
Auth-Application-Id	0	0
Class	0+	0+
Destination-Host	0-1	0
Destination-Realm	1	0
Error-Reporting-Host	0	0+

1.4 Standards Compliance

For the Diameter header, the Command Code namespace allocation has changed. The new allocation rules are as follows:

The Command Code values 256 - 8,388,607 (0x100 to 0x7fffff) are for permanent, standard commands, allocated by IETF Review [RFC5226].

The values 8,388,608 - 16,777,213 (0x800000 - 0xfffffd) are reserved for vendor-specific Command Codes, to be allocated on a First Come, First Served basis by IANA [RFC5226]. The request to IANA for a Vendor-Specific Command Code SHOULD include a reference to a publicly available specification that documents the command in sufficient detail to aid in interoperability between independent implementations. If the specification cannot be made publicly available, the request for a vendor-specific Command Code MUST include the contact information of persons and/or entities responsible for authoring and maintaining the command.

Fajardo, et al. Standards Track [Page 136]

RFC 6733 Diameter Base Protocol October 2012

The values 16,777,214 and 16,777,215 (hexadecimal values 0xfffffe - 0xfffff) are reserved for experimental commands. As these codes are only for experimental and testing purposes, no guarantee is made for interoperability between Diameter peers using experimental commands.

11.2.2. Command Flags

Section 3 describes the existing Command Flags field. The remaining bits can only be assigned via a Standards Action [RFC5226].

11.3. AVP Values

For AVP values, the Experimental-Result-Code AVP value allocation has been added; see Section 11.3.1. The old AVP value allocation rule, IETF Consensus, has been updated to IETF Review as per [RFC5226], and affected AVPs are listed as reminders.

11.3.1. Experimental-Result-Code AVP

Values for this AVP are purely local to the indicated vendor, and no IANA registry is maintained for them.

11.3.2. Result-Code AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.3. Accounting-Record-Type AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.4. Termination-Cause AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.5. Redirect-Host-Usage AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.6. Session-Server-Failover AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.7. Session-Binding AVP Values

New values are available for assignment via IETF Review [RFC5226].

Fajardo, et al. Standards Track [Page 137]

RFC 6733 Diameter Base Protocol October 2012

11.3.8. Disconnect-Cause AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.9. Auth-Request-Type AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.10. Auth-Session-State AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.11. Re-Auth-Request-Type AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.12. Accounting-Realtime-Required AVP Values

New values are available for assignment via IETF Review [RFC5226].

11.3.13. Inband-Security-Id AVP (code 299)

The use of this AVP has been deprecated.

11.4. _diameters Service Name and Port Number Registration

IANA has registered the "_diameters" service name and assigned port numbers for TLS/TCP and DTLS/SCTP according to the guidelines given in [RFC6335].

```

Service Name:      _diameters
Transport Protocols: TCP, SCTP
Assignee:         IESG <iesg@ietf.org>
Contact:         IETF Chair <chair@ietf.org>
Description:      Diameter over TLS/TCP and DTLS/SCTP
Reference:        RFC 6733
Port Number:      5868, from the User Range

```

11.5. SCTP Payload Protocol Identifiers

Two SCTP payload protocol identifiers have been registered in the SCTP Payload Protocol Identifiers registry:

Value	SCTP Payload Protocol Identifier
46	Diameter in a SCTP DATA chunk
47	Diameter in a DTLS/SCTP DATA chunk

11.6. S-NAPTR Parameters

The following tag has been registered in the S-NAPTR Application Protocol Tags registry:

Tag	Protocol
diameter.dtls.sctp	DTLS/SCTP

12. Diameter Protocol-Related Configurable Parameters

This section contains the configurable parameters that are found throughout this document:

Diameter Peer

A Diameter entity MAY communicate with peers that are statically configured. A statically configured Diameter peer would require that either the IP address or the fully qualified domain name (FQDN) be supplied, which would then be used to resolve through DNS.

Routing Table

A Diameter proxy server routes messages based on the realm portion of a Network Access Identifier (NAI). The server MUST have a table of Realm Names, and the address of the peer to which the message must be forwarded. The routing table MAY also include a "default route", which is typically used for all messages that cannot be locally processed.

Tc timer

The Tc timer controls the frequency that transport connection attempts are done to a peer with whom no active transport connection exists. The recommended value is 30 seconds.

13. Security Considerations

The Diameter base protocol messages SHOULD be secured by using TLS [RFC5246] or DTLS/SCTP [RFC6083]. Additional security mechanisms such as IPsec [RFC4301] MAY also be deployed to secure connections between peers. However, all Diameter base protocol implementations MUST support the use of TLS/TCP and DTLS/SCTP, and the Diameter protocol MUST NOT be used without one of TLS, DTLS, or IPsec.

If a Diameter connection is to be protected via TLS/TCP and DTLS/SCTP or IPsec, then TLS/TCP and DTLS/SCTP or IPsec/IKE SHOULD begin prior to any Diameter message exchange. All security parameters for TLS/TCP and DTLS/SCTP or IPsec are configured independent of the Diameter protocol. All Diameter messages will be sent through the TLS/TCP and DTLS/SCTP or IPsec connection after a successful setup.

For TLS/TCP and DTLS/SCTP connections to be established in the open state, the CER/CEA exchange MUST include an Inband-Security-ID AVP with a value of TLS/TCP and DTLS/SCTP. The TLS/TCP and DTLS/SCTP handshake will begin when both ends successfully reach the open state, after completion of the CER/CEA exchange. If the TLS/TCP and DTLS/SCTP handshake is successful, all further messages will be sent via TLS/TCP and DTLS/SCTP. If the handshake fails, both ends MUST move to the closed state. See Section 13.1 for more details.

13.1. TLS/TCP and DTLS/SCTP Usage

Diameter nodes using TLS/TCP and DTLS/SCTP for security MUST mutually authenticate as part of TLS/TCP and DTLS/SCTP session establishment. In order to ensure mutual authentication, the Diameter node acting as the TLS/TCP and DTLS/SCTP server MUST request a certificate from the Diameter node acting as TLS/TCP and DTLS/SCTP client, and the Diameter node acting as the TLS/TCP and DTLS/SCTP client MUST be prepared to supply a certificate on request.

Diameter nodes MUST be able to negotiate the following TLS/TCP and DTLS/SCTP cipher suites:

```
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
```

Diameter nodes SHOULD be able to negotiate the following TLS/TCP and DTLS/SCTP cipher suite:

```
TLS_RSA_WITH_AES_128_CBC_SHA
```

Note that it is quite possible that support for the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite will be REQUIRED at some future date. Diameter nodes MAY negotiate other TLS/TCP and DTLS/SCTP cipher suites.

If public key certificates are used for Diameter security (for example, with TLS), the value of the expiration times in the routing and peer tables MUST NOT be greater than the expiry time in the

successfully compromise the intermediary would imply a high probability of being able to compromise the endpoints as well. Note that no end-to-end security mechanism is specified in this document.

14. References

14.1. Normative References

[FLOATPOINT]

Institute of Electrical and Electronics Engineers, "IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985", August 1985.

[IANAADFAM]

IANA, "Address Family Numbers",
<<http://www.iana.org/assignments/address-family-numbers>>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

Fajardo, et al. Standards Track [Page 142]

RFC 6733 Diameter Base Protocol October 2012

[RFC3539] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, June 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4004] Calhoun, P., Johansson, T., Perkins, C., Hiller, T., and P. McCann, "Diameter Mobile IPv4 Application", RFC 4004, August 2005.

[RFC4005] Calhoun, P., Zorn, G., Spence, D., and D. Mitton, "Diameter Network Access Server Application", RFC 4005, August 2005.

[RFC4006] Hakala, H., Mattila, L., Koskinen, J-P., Stura, M., and J. Loughney, "Diameter Credit-Control Application", RFC 4006, August 2005.

aaa-parameters.xml#aaa-parameters-16>.

- [RFC1492] Finseth, C., "An Access Control Protocol, Sometimes Called TACACS", RFC 1492, July 1993.
- [RFC1661] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

Fajardo, et al. Standards Track [Page 144]

RFC 6733 Diameter Base Protocol October 2012

- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC2869] Rigney, C., Willats, W., and P. Calhoun, "RADIUS Extensions", RFC 2869, June 2000.
- [RFC2881] Mitton, D. and M. Beadles, "Network Access Server Requirements Next Generation (NASREQNG) NAS Model", RFC 2881, July 2000.
- [RFC2975] Aboba, B., Arkko, J., and D. Harrington, "Introduction to Accounting Management", RFC 2975, October 2000.
- [RFC2989] Aboba, B., Calhoun, P., Glass, S., Hiller, T., McCann, P., Shiino, H., Walsh, P., Zorn, G., Dommety, G., Perkins, C., Patil, B., Mitton, D., Manning, S., Beadles, M., Chen, X., Sivalingham, S., Hameed, A., Munson, M., Jacobs, S., Lim, B., Hirschman, B., Hsu, R., Koo, H., Lipford, M., Campbell, E., Xu, Y., Baba, S., and E. Jaques, "Criteria for Evaluating AAA Protocols for Network Access", RFC 2989, November 2000.
- [RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.
- [RFC5176] Chiba, M., Dommety, G., Eklund, M., Mitton, D., and B.

1.4 Standards Compliance

Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, January 2008.

Fajardo, et al. Standards Track [Page 145]

RFC 6733 Diameter Base Protocol October 2012

[RFC5461] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, February 2009.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[RFC6737] Kang, J. and G. Zorn, "The Diameter Capabilities Update Application", RFC 6737, October 2012.

Fajardo, et al. Standards Track [Page 146]

Appendix A. Acknowledgements

A.1. This Document

The authors would like to thank the following people that have provided proposals and contributions to this document:

To Vishnu Ram and Satendra Gera for their contributions on capabilities updates, predictive loop avoidance, as well as many other technical proposals. To Tolga Asveren for his insights and contributions on almost all of the proposed solutions incorporated into this document. To Timothy Smith for helping on the capabilities Update and other topics. To Tony Zhang for providing fixes to loopholes on composing Failed-AVPs as well as many other issues and topics. To Jan Nordqvist for clearly stating the usage of Application Ids. To Anders Kristensen for providing needed technical opinions. To David Frascione for providing invaluable review of the document. To Mark Jones for providing clarifying text on vendor command codes and other vendor-specific indicators. To Victor Pascual and Sebastien Decugis for new text and recommendations on SCTP/DTLS. To Jouni Korhonen for taking over the editing task and resolving last bits from versions 27 through 29.

Special thanks to the Diameter extensibility design team, which helped resolve the tricky question of mandatory AVPs and ABNF semantics. The members of this team are as follows:

Avi Lior, Jari Arkko, Glen Zorn, Lionel Morand, Mark Jones, Tolga Asveren, Jouni Korhonen, and Glenn McGregor.

Special thanks also to people who have provided invaluable comments and inputs especially in resolving controversial issues:

Glen Zorn, Yoshihiro Ohba, Marco Stura, Stephen Farrel, Pete Resnick, Peter Saint-Andre, Robert Sparks, Krishna Prasad, Sean Turner, Barry Leiba, and Pasi Eronen.

Finally, we would like to thank the original authors of this document:

Pat Calhoun, John Loughney, Jari Arkko, Erik Guttman, and Glen Zorn.

Their invaluable knowledge and experience has given us a robust and flexible AAA protocol that many people have seen great value in adopting. We greatly appreciate their support and stewardship for the continued improvements of Diameter as a protocol. We would also like to extend our gratitude to folks aside from the authors who have

assisted and contributed to the original version of this document. Their efforts significantly contributed to the success of Diameter.

A.2. RFC 3588

The authors would like to thank Nenad Trifunovic, Tony Johansson and

1.4 Standards Compliance

Pankaj Patel for their participation in the pre-IETF Document Reading Party. Allison Mankin, Jonathan Wood, and Bernard Aboba provided invaluable assistance in working out transport issues and this was also the case with Steven Bellovin in the security area.

Paul Funk and David Mitton were instrumental in getting the Peer State Machine correct, and our deep thanks go to them for their time.

Text in this document was also provided by Paul Funk, Mark Eklund, Mark Jones, and Dave Spence. Jacques Caron provided many great comments as a result of a thorough review of the spec.

The authors would also like to acknowledge the following people for their contribution in the development of the Diameter protocol:

Allan C. Rubens, Haseeb Akhtar, William Bulley, Stephen Farrell, David Frascione, Daniel C. Fox, Lol Grant, Ignacio Goyret, Nancy Greene, Peter Heitman, Fredrik Johansson, Mark Jones, Martin Julien, Bob Kopacz, Paul Krumviede, Fergal Ladley, Ryan Moats, Victor Muslin, Kenneth Peirce, John Schnizlein, Sumit Vakil, John R. Vollbrecht, and Jeff Weisberg.

Finally, Pat Calhoun would like to thank Sun Microsystems since most of the effort put into this document was done while he was in their employ.

Appendix B. S-NAPTR Example

As an example, consider a client that wishes to resolve `aaa:ex1.example.com`. The client performs a NAPTR query for that domain, and the following NAPTR records are returned:

```
;;      order pref flags service  regexp replacement
IN NAPTR 50 50 "s" "aaa:diameter.tls.tcp" ""
        _diameter._tls.ex1.example.com
IN NAPTR 100 50 "s" "aaa:diameter.tcp" ""
        _aaa._tcp.ex1.example.com
IN NAPTR 150 50 "s" "aaa:diameter.sctp" ""
        _diameter._sctp.ex1.example.com
```

This indicates that the server supports TLS, TCP, and SCTP in that order. If the client supports TLS, TLS will be used, targeted to a

Fajardo, et al. Standards Track [Page 148]

RFC 6733 Diameter Base Protocol October 2012

host determined by an SRV lookup of `_diameter._tls.ex1.example.com`. That lookup would return:

```
;;      Priority Weight Port Target
IN SRV 0 1 5060 server1.ex1.example.com
IN SRV 0 2 5060 server2.ex1.example.com
```

As an alternative example, a client that wishes to resolve `aaa:ex2.example.com`. The client performs a NAPTR query for that domain, and the following NAPTR records are returned:

```
;;      order pref flags service  regexp replacement
IN NAPTR 150 50 "a" "aaa:diameter.tls.tcp" ""
        server1.ex2.example.com
IN NAPTR 150 50 "a" "aaa:diameter.tls.tcp" ""
```

server2.ex2.example.com

This indicates that the server supports TCP available at the returned host names.

Appendix C. Duplicate Detection

As described in Section 9.4, accounting record duplicate detection is based on session identifiers. Duplicates can appear for various reasons:

- o Failover to an alternate server. Where close to real-time performance is required, failover thresholds need to be kept low. This may lead to an increased likelihood of duplicates. Failover can occur at the client or within Diameter agents.
- o Failure of a client or agent after sending a record from non-volatile memory, but prior to receipt of an application-layer ACK and deletion of the record to be sent. This will result in retransmission of the record soon after the client or agent has rebooted.
- o Duplicates received from RADIUS gateways. Since the retransmission behavior of RADIUS is not defined within [RFC2865], the likelihood of duplication will vary according to the implementation.
- o Implementation problems and misconfiguration.

The T flag is used as an indication of an application-layer retransmission event, e.g., due to failover to an alternate server. It is defined only for request messages sent by Diameter clients or agents. For instance, after a reboot, a client may not know whether

Fajardo, et al.

Standards Track

[Page 149]

RFC 6733

Diameter Base Protocol

October 2012

it has already tried to send the accounting records in its non-volatile memory before the reboot occurred. Diameter servers MAY use the T flag as an aid when processing requests and detecting duplicate messages. However, servers that do this MUST ensure that duplicates are found even when the first transmitted request arrives at the server after the retransmitted request. It can be used only in cases where no answer has been received from the server for a request and the request is sent again, (e.g., due to a failover to an alternate peer, due to a recovered primary peer or due to a client re-sending a stored record from non-volatile memory such as after reboot of a client or agent).

In some cases, the Diameter accounting server can delay the duplicate detection and accounting record processing until a post-processing phase takes place. At that time records are likely to be sorted according to the included User-Name and duplicate elimination is easy in this case. In other situations, it may be necessary to perform real-time duplicate detection, such as when credit limits are imposed or real-time fraud detection is desired.

In general, only generation of duplicates due to failover or re-sending of records in non-volatile storage can be reliably detected by Diameter clients or agents. In such cases, the Diameter client or agents can mark the message as a possible duplicate by setting the T

1.4 Standards Compliance

flag. Since the Diameter server is responsible for duplicate detection, it can choose whether or not to make use of the T flag, in order to optimize duplicate detection. Since the T flag does not affect interoperability, and it may not be needed by some servers, generation of the T flag is REQUIRED for Diameter clients and agents, but it MAY be implemented by Diameter servers.

As an example, it can be usually be assumed that duplicates appear within a time window of longest recorded network partition or device fault, perhaps a day. So only records within this time window need to be looked at in the backward direction. Secondly, hashing techniques or other schemes, such as the use of the T flag in the received messages, may be used to eliminate the need to do a full search even in this set except for rare cases.

The following is an example of how the T flag may be used by the server to detect duplicate requests.

A Diameter server MAY check the T flag of the received message to determine if the record is a possible duplicate. If the T flag is set in the request message, the server searches for a duplicate within a configurable duplication time window backward and forward. This limits database searching to those records where the T flag is set. In a well-run network, network partitions and

Fajardo, et al. Standards Track [Page 150]
RFC 6733 Diameter Base Protocol October 2012

device faults will presumably be rare events, so this approach represents a substantial optimization of the duplicate detection process. During failover, it is possible for the original record to be received after the T-flag-marked record, due to differences in network delays experienced along the path by the original and duplicate transmissions. The likelihood of this occurring increases as the failover interval is decreased. In order to be able to detect duplicates that are out of order, the Diameter server should use backward and forward time windows when performing duplicate checking for the T-flag-marked request. For example, in order to allow time for the original record to exit the network and be recorded by the accounting server, the Diameter server can delay processing records with the T flag set until a time period `TIME_WAIT + RECORD_PROCESSING_TIME` has elapsed after the closing of the original transport connection. After this time period, it may check the T-flag-marked records against the database with relative assurance that the original records, if sent, have been received and recorded.

Appendix D. Internationalized Domain Names

To be compatible with the existing DNS infrastructure and simplify host and domain name comparison, Diameter identities (FQDNs) are represented in ASCII form. This allows the Diameter protocol to fall in-line with the DNS strategy of being transparent from the effects of Internationalized Domain Names (IDNs) by following the recommendations in [RFC4690] and [RFC5890]. Applications that provide support for IDNs outside of the Diameter protocol but interacting with it SHOULD use the representation and conversion framework described in [RFC5890], [RFC5891], and [RFC3492].

1.4.2 RFC 3539

RFC 3539 is more difficult to comply to since it discusses problems as much as it requires functionality but all the MUST's are covered, the watchdog state machine being the primary one. Of the optional functionality, load balancing is left to the diameter user (since it's the one deciding who to send to) and there is no Congestion Manager.

2 Reference Manual

The Diameter application is a framework for building applications on top of the Diameter protocol.

diameter

Erlang module

This module provides the interface with which a user can implement a Diameter node that sends and receives messages using the Diameter protocol as defined in RFC 6733.

Basic usage consists of creating a representation of a locally implemented Diameter node and its capabilities with *start_service/2*, adding transport capability using *add_transport/2* and sending Diameter requests and receiving Diameter answers with *call/4*. Incoming Diameter requests are communicated as callbacks to a *diameter_app(3)* callback modules as specified in the service configuration.

Beware the difference between **diameter** (not capitalized) and **Diameter** (capitalized). The former refers to the Erlang application named diameter whose main api is defined here, the latter to Diameter protocol in the sense of RFC 6733.

The diameter application must be started before calling most functions in this module.

DATA TYPES

Address()
 DiameterIdentity()
 Grouped()
 OctetString()
 Time()
 Unsigned32()
 UTF8String()

Types corresponding to RFC 6733 AVP Data Formats. Defined in *diameter_dict(4)*.

application_alias() = term()

Name identifying a Diameter application in service configuration. Passed to *call/4* when sending requests defined by the application.

application_module() = Mod | [Mod | ExtraArgs] | #diameter_callback{}

```
Mod = atom()
ExtraArgs = list()
```

Module implementing the callback interface defined in *diameter_app(3)*, along with any extra arguments to be appended to those documented. Note that extra arguments specific to an outgoing request can be specified to *call/4*, in which case those are appended to any module-specific extra arguments.

Specifying a #diameter_callback{} record allows individual functions to be configured in place of the usual *diameter_app(3)* callbacks. See *diameter_callback.erl* for details.

application_opt()

Options defining a Diameter application. Has one the following types.

```
{alias, application_alias()}
```

Unique identifier for the application in the scope of the service. Defaults to the value of the dictionary option.

{dictionary, atom()}

Name of an encode/decode module for the Diameter messages defined by the application. These modules are generated from files whose format is documented in *diameter_dict(4)*.

{module, application_module()}

Callback module in which messages of the Diameter application are handled. See *diameter_app(3)* for the required interface and semantics.

{state, term()}

Initial callback state. The prevailing state is passed to some *diameter_app(3)* callbacks, which can then return a new state. Defaults to the value of the *alias* option.

{call_mutates_state, true|false}

Whether or not the *pick_peer/4* application callback can modify the application state. Defaults to *false*.

Warning:

pick_peer/4 callbacks are serialized when this option is *true*, which is a potential performance bottleneck. A simple Diameter client may suffer no ill effects from using mutable state but a server or agent that responds to incoming request should probably avoid it.

{answer_errors, callback|report|discard}

Manner in which incoming answer messages containing decode errors are handled.

If *callback* then errors result in a *handle_answer/4* callback in the same fashion as for *handle_request/3*, with errors communicated in the *errors* field of the `#diameter_packet{ }` passed to the callback. If *report* then an answer containing errors is discarded without a callback and a warning report is written to the log. If *discard* then an answer containing errors is silently discarded without a callback. In both the *report* and *discard* cases the return value for the *call/4* invocation in question is as if a callback had taken place and returned `{error, failure}`.

Defaults to *discard*.

{request_errors, answer_3xxx|answer|callback}

Manner in which incoming requests are handled when an error other than 3007 (DIAMETER_APPLICATION_UNSUPPORTED, which cannot be associated with an application callback module), is detected.

If *answer_3xxx* then requests are answered without a *handle_request/3* callback taking place. If *answer* then even 5xxx errors are answered without a callback unless the connection in question has configured the RFC 3588 common dictionary as noted below. If *callback* then a *handle_request/3* callback always takes place and its return value determines the answer sent to the peer, if any.

Defaults to *answer_3xxx*.

Note:

Answers sent by diameter set the E-bit in the Diameter Header. Since RFC 3588 allows only 3xxx result codes in an `answer-message`, `answer` has the same semantics as `answer_3xxx` when the transport in question has been configured with `diameter_gen_base_rfc3588` as its common dictionary. Since RFC 6733 allows both 3xxx and 5xxx result codes in an `answer-message`, a transport with `diameter_gen_base_rfc6733` as its common dictionary does distinguish between `answer_3xxx` and `answer`.

`call_opt()`

Options available to `call/4` when sending an outgoing Diameter request. Has one of the following types.

```
{extra, list()}
```

Extra arguments to append to callbacks to the callback module in question. These are appended to any extra arguments configured on the callback itself. Multiple options append to the argument list.

```
{filter, peer_filter()}
```

Filter to apply to the list of available peers before passing it to the `pick_peer/4` callback for the application in question. Multiple options are equivalent a single `all` filter on the corresponding list of filters. Defaults to `none`.

```
{timeout, Unsigned32()}
```

Number of milliseconds after which the request should timeout. Defaults to 5000.

`detach`

Cause `call/4` to return `ok` as soon as the request in question has been encoded, instead of waiting for and returning the result from a subsequent `handle_answer/4` or `handle_error/4` callback.

An invalid option will cause `call/4` to fail.

`capability()`

AVP values sent in outgoing CER or CEA messages during capabilities exchange. Can be configured both on a service and a transport, values on the latter taking precedence. Has one of the following types.

```
{'Origin-Host', DiameterIdentity()}
{'Origin-Realm', DiameterIdentity()}
{'Host-IP-Address', [Address()]}
```

An address list is available to the start function of a `transport module`, which can return a new list for use in the subsequent CER or CEA. `Host-IP-Address` need not be specified if the transport module in question communicates an address list as described in `diameter_transport(3)`

```
{'Vendor-Id', Unsigned32()}
{'Product-Name', UTF8String()}
{'Origin-State-Id', Unsigned32()}
```

`Origin-State-Id` is optional but, if configured, will be included in outgoing CER/CEA and DWR/DWA messages. Setting a value of 0 (zero) is equivalent to not setting a value, as documented in RFC 6733. The function `origin_state_id/0` can be used as to retrieve a value that is computed when the diameter application is started.

```
{'Supported-Vendor-Id', [Unsigned32()]}
{'Auth-Application-Id', [Unsigned32()]}
```

```
{'Inband-Security-Id', [Unsigned32()]}
```

Inband-Security-Id defaults to the empty list, which is equivalent to a list containing only 0 (NO_INBAND_SECURITY). If 1 (TLS) is specified then TLS is selected if the CER/CEA received from the peer offers it.

```
{'Acct-Application-Id', [Unsigned32()]}
{'Vendor-Specific-Application-Id', [Grouped()]}
{'Firmware-Revision', Unsigned32()}
```

Note that each tuple communicates one or more AVP values. It is an error to specify duplicate tuples.

```
evaluable() = {M,F,A} | fun() | [evaluable() | A]
```

An expression that can be evaluated as a function in the following sense.

```
eval([M,F,A] | T) ->
  apply(M, F, T ++ A);
eval([F|A] | T) ->
  eval([F | T ++ A]);
eval([F|A]) ->
  apply(F, A);
eval(F) ->
  eval([F]).
```

Applying an *evaluable()* E to an argument list A is meant in the sense of `eval([E|A])`.

Warning:

Beware of using fun expressions of the form `fun Name/Arity` in situations in which the fun is not short-lived and code is to be upgraded at runtime since any processes retaining such a fun will have a reference to old code. In particular, such a value is typically inappropriate in configuration passed to *start_service/2* or *add_transport/2*.

```
peer_filter() = term()
```

Filter passed to *call/4* in order to select candidate peers for a *pick_peer/4* callback. Has one of the following types.

none

Matches any peer. This is a convenience that provides a filter equivalent to no filter.

host

Matches only those peers whose Origin-Host has the same value as Destination-Host in the outgoing request in question, or any peer if the request does not contain a Destination-Host AVP.

realm

Matches only those peers whose Origin-Realm has the same value as Destination-Realm in the outgoing request in question, or any peer if the request does not contain a Destination-Realm AVP.

```
{host, any|DiameterIdentity()}
```

Matches only those peers whose Origin-Host has the specified value, or all peers if the atom any.

```
{realm, any|DiameterIdentity()}
```

Matches only those peers whose Origin-Realm has the specified value, or all peers if the atom any.

```
{eval, evaluable() }
```

Matches only those peers for which the specified *evaluable()* returns true when applied to the connection's *diameter_caps* record. Any other return value or exception is equivalent to false.

```
{neg, peer_filter() }
```

Matches only those peers not matched by the specified filter.

```
{all, [peer_filter() ] }
```

Matches only those peers matched by each filter in the specified list.

```
{any, [peer_filter() ] }
```

Matches only those peers matched by at least one filter in the specified list. The resulting list will be in match order, peers matching the first filter of the list sorting before those matched by the second, and so on.

```
{first, [peer_filter() ] }
```

Like *any*, but stops at the first filter for which there are matches, which can be much more efficient when there are many peers. For example, the following filter causes only peers best matching both the host and realm filters to be presented.

```
{first, [{all, [host, realm]}, realm]}
```

An invalid filter is equivalent to `{any, []}`, a filter that matches no peer.

Note:

The *host* and *realm* filters cause the Destination-Host and Destination-Realm AVPs to be extracted from the outgoing request, assuming it to be a record- or list-valued *diameter_codec:message()*, and assuming at most one of each AVP. If this is not the case then the `{host|realm, DiameterIdentity() }` filters must be used to achieve the desired result. An empty *DiameterIdentity()* (which should not be typical) matches all hosts/realm for the purposes of filtering.

Warning:

A *host* filter is not typically desirable when setting Destination-Host since it will remove peer agents from the candidates list.

```
service_event() = #diameter_event{service = service_name(), info = service_event_info() }
```

An event message sent to processes that have subscribed to these using *subscribe/1*.

```
service_event_info() = term()
```

The *info* field of a *service_event()* record. Can have one of the following types.

```
start
```

stop

The service is being started or stopped. No event precedes a `start` event. No event follows a `stop` event, and this event implies the termination of all transport processes.

```
{up, Ref, Peer, Config, Pkt}  
{up, Ref, Peer, Config}  
{down, Ref, Peer, Config}
```

```
Ref    = transport_ref()  
Peer   = diameter_app:peer()  
Config = {connect|listen, [transport_opt()]}  
Pkt    = #diameter_packet{}
```

The RFC 3539 watchdog state machine has transitioned into (up) or out of (down) the OKAY state. If a `#diameter_packet{}` is present in an up event then there has been a capabilities exchange on a newly established transport connection and the record contains the received CER or CEA.

Note that a single up or down event for a given peer corresponds to multiple `peer_up/3` or `peer_down/3` callbacks, one for each of the Diameter applications negotiated during capabilities exchange. That is, the event communicates connectivity with the peer as a whole while the callbacks communicate connectivity with respect to individual Diameter applications.

```
{reconnect, Ref, Opts}
```

```
Ref = transport_ref()  
Opts = [transport_opt()]
```

A connecting transport is attempting to establish/reestablish a transport connection with a peer following `connect_timer` or `watchdog_timer` expiry.

```
{closed, Ref, Reason, Config}
```

```
Ref = transport_ref()  
Config = {connect|listen, [transport_opt()]}
```

Capabilities exchange has failed. Reason can have one of the following types.

```
{'CER', Result, Caps, Pkt}
```

```
Result = ResultCode | {capabilities_cb, CB, ResultCode|discard}  
Caps = #diameter_caps{  
Pkt = #diameter_packet{  
ResultCode = integer()  
CB = evaluable()
```

An incoming CER has been answered with the indicated result code, or discarded. `Caps` contains pairs of values, for the local node and remote peer respectively. `Pkt` contains the CER in question. In the case of rejection by a capabilities callback, the tuple contains the rejecting callback.

```
{'CER', Caps, {ResultCode, Pkt}}
```

```

ResultCode = integer()
Caps = #diameter_caps{}
Pkt = #diameter_packet{}

```

An incoming CER contained errors and has been answered with the indicated result code. `Caps` contains values for the local node only. `Pkt` contains the CER in question.

```
{'CER', timeout}
```

An expected CER was not received within `capx_timeout` of connection establishment.

```
{'CEA', Result, Caps, Pkt}
```

```

Result = ResultCode | atom() | {capabilities_cb, CB, ResultCode|discard}
Caps = #diameter_caps{}
Pkt = #diameter_packet{}
ResultCode = integer()

```

An incoming CEA has been rejected for the indicated reason. An integer-valued `Result` indicates the result code sent by the peer. `Caps` contains pairs of values for the local node and remote peer. `Pkt` contains the CEA in question. In the case of rejection by a capabilities callback, the tuple contains the rejecting callback.

```
{'CEA', Caps, Pkt}
```

```

Caps = #diameter_caps{}
Pkt = #diameter_packet{}

```

An incoming CEA contained errors and has been rejected. `Caps` contains only values for the local node. `Pkt` contains the CEA in question.

```
{'CEA', timeout}
```

An expected CEA was not received within `capx_timeout` of connection establishment.

```
{watchdog, Ref, PeerRef, {From, To}, Config}
```

```

Ref = transport_ref()
PeerRef = diameter_app:peer_ref()
From, To = initial | okay | suspect | down | reopen
Config = {connect|listen, [transport_opt()]}

```

An RFC 3539 watchdog state machine has changed state.

```
any()
```

For forward compatibility, a subscriber should be prepared to receive info fields of forms other than the above.

```
service_name() = term()
```

Name of a service as passed to `start_service/2` and with which the service is identified. There can be at most one service with a given name on a given node. Note that `erlang:make_ref/0` can be used to generate a service name that is somewhat unique.

`service_opt()`

Option passed to `start_service/2`. Can be any `capability()` as well as the following.

`{application, [application_opt()]}`

A Diameter application supported by the service.

A service must configure one tuple for each Diameter application it intends to support. For an outgoing request, the relevant `application_alias()` is passed to `call/4`, while for an incoming request the application identifier in the message header determines the application, the identifier being specified in the application's `dictionary` file.

Warning:

The capabilities advertised by a node must match its configured applications. In particular, application configuration must be matched by corresponding `capability()` configuration, of *-Application-Id AVPs in particular.

`{incoming_maxlen, 0..16777215}`

Bound on the expected size of incoming Diameter messages. Messages larger than the specified number of bytes are discarded.

Defaults to 16777215, the maximum value of the 24-bit Message Length field in a Diameter Header.

`{restrict_connections, false | node | nodes | [node()] | evaluable()}`

The degree to which the service allows multiple transport connections to the same peer, as identified by its Origin-Host at capabilities exchange.

If `[node()]` then a connection is rejected if another already exists on any of the specified nodes. Types `false`, `node`, `nodes` and `evaluable()` are equivalent to `[]`, `[node()]`, `[node() | nodes()]` and the evaluated value respectively, evaluation of each expression taking place whenever a new connection is to be established. Note that `false` allows an unlimited number of connections to be established with the same peer.

Multiple connections are independent and governed by their own peer and watchdog state machines.

Defaults to `nodes`.

`{sequence, {H,N} | evaluable()}`

A constant value `H` for the topmost `32-N` bits of of 32-bit End-to-End and Hop-by-Hop Identifiers generated by the service, either explicitly or as a return value of a function to be evaluated at `start_service/2`. In particular, an identifier `Id` is mapped to a new identifier as follows.

```
(H bsl N) bor (Id band ((1 bsl N) - 1))
```

Note that RFC 6733 requires that End-to-End Identifiers remain unique for a period of at least 4 minutes and that this and the call rate places a lower bound on appropriate values of `N`: at a rate of `R` requests per second, an `N`-bit counter traverses all of its values in $(1 \text{ bsl } N) \text{ div } (R * 60)$ minutes, so the bound is $4 * R * 60 \leq 1 \text{ bsl } N$.

`N` must lie in the range `0..32` and `H` must be a non-negative integer less than $1 \text{ bsl } (32 - N)$.

Defaults to `{0, 32}`.

Warning:

Multiple Erlang nodes implementing the same Diameter node should be configured with different sequence masks to ensure that each node uses a unique range of End-to-End and Hop-by-Hop Identifiers for outgoing requests.

```
{share_peers, boolean() | [node()] | evaluable() }
```

Nodes to which peer connections established on the local Erlang node are communicated. Shared peers become available in the remote candidates list passed to *pick_peer/4* callbacks on remote nodes whose services are configured to use them: see *use_shared_peers* below.

If *false* then peers are not shared. If *[node()]* then peers are shared with the specified list of nodes. If *evaluable()* then peers are shared with the nodes returned by the specified function, evaluated whenever a peer connection becomes available or a remote service requests information about local connections. The value *true* is equivalent to *fun erlang:nodes/0*. The value *node()* in a list is ignored, so a collection of services can all be configured to share with the same list of nodes.

Defaults to *false*.

Note:

Peers are only shared with services of the same name for the purpose of sending outgoing requests. Since the value of the *application_opt()* *alias*, passed to *call/4*, is the handle for identifying a peer as a suitable candidate, services that share peers must use the same aliases to identify their supported applications. They should typically also configure identical *capabilities()*, since by sharing peer connections they are distributing the implementation of a single Diameter node across multiple Erlang nodes.

```
{spawn_opt, [term()] }
```

Options list passed to *erlang:spawn_opt/2* when spawning a process for an incoming Diameter request, unless the transport in question specifies another value. Options *monitor* and *link* are ignored.

Defaults to the empty list.

```
{strict_mbit, boolean() }
```

Whether or not to regard an AVP setting the M-bit as erroneous when the command grammar in question does not explicitly allow the AVP. If *true* then such AVPs are regarded as 5001 errors, *DIAMETER_AVP_UNSUPPORTED*. If *false* then the M-bit is ignored and policing it becomes the receiver's responsibility.

Defaults to *true*.

Warning:

RFC 6733 is unclear about the semantics of the M-bit. On the one hand, the CCF specification in section 3.2 documents AVP in a command grammar as meaning **any** arbitrary AVP; on the other hand, 1.3.4 states that AVPs setting the M-bit cannot be added to an existing command: the modified command must instead be placed in a new Diameter application.

The reason for the latter is presumably interoperability: allowing arbitrary AVPs setting the M-bit in a command makes its interpretation implementation-dependent, since there's no guarantee that all implementations will understand the same set of arbitrary AVPs in the context of a given command. However, interpreting AVP in a command grammar as any AVP, regardless of M-bit, renders 1.3.4 meaningless, since the receiver can simply ignore any AVP it thinks isn't relevant, regardless of the sender's intent.

Beware of confusing mandatory in the sense of the M-bit with mandatory in the sense of the command grammar. The former is a semantic requirement: that the receiver understand the semantics of the AVP in the context in question. The latter is a syntactic requirement: whether or not the AVP must occur in the message in question.

```
{string_decode, boolean() }
```

Whether or not to decode AVPs of type *OctetString()* and its derived types *DiameterIdentity()*, *DiameterURI()*, *IPFilterRule()*, *QoSFilterRule()*, and *UTF8String()*. If `true` then AVPs of these types are decoded to string(). If `false` then values are retained as binary().

Defaults to `true`.

Warning:

This option should be set to `false` since a sufficiently malicious peer can otherwise cause large amounts of memory to be consumed when decoded Diameter messages are passed between processes. The default value is for backwards compatibility.

```
{use_shared_peers, boolean() | [node()] | evaluable() }
```

Nodes from which communicated peers are made available in the remote candidates list of *pick_peer/4* callbacks.

If `false` then remote peers are not used. If `[node()]` then only peers from the specified list of nodes are used. If `evaluable()` then only peers returned by the specified function are used, evaluated whenever a remote service communicates information about an available peer connection. The value `true` is equivalent to fun `erlang:nodes/0`. The value `node()` in a list is ignored.

Defaults to `false`.

Note:

A service that does not use shared peers will always pass the empty list as the second argument of *pick_peer/4* callbacks.

Warning:

Sending a request over a peer connection on a remote node is less efficient than sending it over a local connection. It may be preferable to make use of the *service_opt()* *restrict_connections* and maintain a dedicated connection on each node from which requests are sent.

`transport_opt()`

Option passed to *add_transport/2*. Has one of the following types.

`{applications, [application_alias()]}`

Diameter applications to which the transport should be restricted. Defaults to all applications configured on the service in question. Applications not configured on the service in question are ignored.

Warning:

The capabilities advertised by a node must match its configured applications. In particular, setting *applications* on a transport typically implies having to set matching *-Application-Id AVPs in a *capabilities()* tuple.

`{capabilities, [capability()]}`

AVPs used to construct outgoing CER/CEA messages. Values take precedence over any specified on the service in question.

Specifying a capability as a transport option may be particularly appropriate for Inband-Security-Id, in case TLS is desired over TCP as implemented by *diameter_tcp(3)*.

`{capabilities_cb, evaluable()}`

Callback invoked upon reception of CER/CEA during capabilities exchange in order to ask whether or not the connection should be accepted. Applied to the *transport_ref()* and `#diameter_caps{}` record of the connection.

The return value can have one of the following types.

`ok`

Accept the connection.

`integer()`

Causes an incoming CER to be answered with the specified Result-Code.

`discard`

Causes an incoming CER to be discarded without CEA being sent.

`unknown`

Equivalent to returning 3010, `DIAMETER_UNKNOWN_PEER`.

Returning anything but `ok` or a 2xxx series result code causes the transport connection to be broken. Multiple *capabilities_cb* options can be specified, in which case the corresponding callbacks are applied until either all return `ok` or one does not.

```
{capx_timeout, Unsigned32() }
```

Number of milliseconds after which a transport process having an established transport connection will be terminated if the expected capabilities exchange message (CER or CEA) is not received from the peer. For a connecting transport, the timing of connection attempts is governed by *connect_timer* or *watchdog_timer* expiry. For a listening transport, the peer determines the timing.

Defaults to 10000.

```
{connect_timer, Tc }
```

```
Tc = Unsigned32()
```

For a connecting transport, the RFC 6733 Tc timer, in milliseconds. This timer determines the frequency with which a transport attempts to establish an initial connection with its peer following transport configuration. Once an initial connection has been established, *watchdog_timer* determines the frequency of reconnection attempts, as required by RFC 3539.

For a listening transport, the timer specifies the time after which a previously connected peer will be forgotten: a connection after this time is regarded as an initial connection rather than reestablishment, causing the RFC 3539 state machine to pass to state OKAY rather than REOPEN. Note that these semantics are not governed by the RFC and that a listening transport's *connect_timer* should be greater than its peer's Tw plus jitter.

Defaults to 30000 for a connecting transport and 60000 for a listening transport.

```
{disconnect_cb, evaluable() }
```

Callback invoked prior to terminating the transport process of a transport connection having watchdog state OKAY. Applied to *application|service|transport* and the *transport_ref()* and *diameter_app:peer()* in question: *application* indicates that the diameter application is being stopped, *service* that the service in question is being stopped by *stop_service/1*, and *transport* that the transport in question is being removed by *remove_transport/2*.

The return value can have one of the following types.

```
{dpr, [option() ] }
```

Send Disconnect-Peer-Request to the peer, the transport process being terminated following reception of Disconnect-Peer-Answer or timeout. An *option()* can be one of the following.

```
{cause, 0|rebooting|1|busy|2|goaway }
```

Disconnect-Cause to send, REBOOTING, BUSY and DO_NOT_WANT_TO_TALK_TO_YOU respectively. Defaults to *rebooting* for Reason=*service|application* and *goaway* for Reason=*transport*.

```
{timeout, Unsigned32() }
```

Number of milliseconds after which the transport process is terminated if DPA has not been received. Defaults to the value of *dpa_timeout*.

dpr

Equivalent to `{dpr, []}`.

close

Terminate the transport process without Disconnect-Peer-Request being sent to the peer.

ignore

Equivalent to not having configured the callback.

Multiple *disconnect_cb* options can be specified, in which case the corresponding callbacks are applied until one of them returns a value other than `ignore`. All callbacks returning `ignore` is equivalent to not having configured them.

Defaults to a single callback returning `dpr`.

```
{dpa_timeout, Unsigned32() }
```

Number of milliseconds after which a transport connection is terminated following an outgoing DPR if DPA is not received.

Defaults to 1000.

```
{dpr_timeout, Unsigned32() }
```

Number of milliseconds after which a transport connection is terminated following an incoming DPR if the peer does not close the connection.

Defaults to 5000.

```
{length_errors, exit|handle|discard}
```

How to deal with errors in the Message Length field of the Diameter Header in an incoming message. An error in this context is that the length is not at least 20 bytes (the length of a Header), is not a multiple of 4 (a valid length) or is not the length of the message in question, as received over the transport interface documented in *diameter_transport(3)*.

If `exit` then the transport process in question exits. If `handle` then the message is processed as usual, a resulting *handle_request/3* or *handle_answer/4* callback (if one takes place) indicating the 5015 error (DIAMETER_INVALID_MESSAGE_LENGTH). If `discard` then the message in question is silently discarded.

Defaults to `exit`.

Note:

The default value reflects the fact that a transport module for a stream-oriented transport like TCP may not be able to recover from a message length error since such a transport must use the Message Length header to divide the incoming byte stream into individual Diameter messages. An invalid length leaves it with no reliable way to rediscover message boundaries, which may result in the failure of subsequent messages. See *diameter_tcp(3)* for the behaviour of that module.

```
{pool_size, pos_integer() }
```

Number of transport processes to start. For a listening transport, determines the size of the pool of accepting transport processes, a larger number being desirable for processing multiple concurrent peer connection attempts. For a connecting transport, determines the number of connections to the peer in question that will be attempted to be established: the *service_opt(): restrict_connections* should also be configured on the service in question to allow multiple connections to the same peer.

```
{spawn_opt, [term() ] }
```

Options passed to *erlang:spawn_opt/2* when spawning a process for an incoming Diameter request. Options `monitor` and `link` are ignored.

Defaults to the list configured on the service if not specified.

```
{transport_config, term()}
{transport_config, term(), Unsigned32() | infinity}
```

Term passed as the third argument to the *start/3* function of the relevant *transport module* in order to start a transport process. Defaults to the empty list.

The 3-tuple form additionally specifies an interval, in milliseconds, after which a started transport process should be terminated if it has not yet established a connection. For example, the following options on a connecting transport request a connection with one peer over SCTP or another (typically the same) over TCP.

```
{transport_module, diameter_sctp}
{transport_config, SctpOpts, 5000}
{transport_module, diameter_tcp}
{transport_config, TcpOpts}
```

To listen on both SCTP and TCP, define one transport for each.

```
{transport_module, atom()}
```

Module implementing a transport process as defined in *diameter_transport(3)*. Defaults to *diameter_tcp*.

Multiple *transport_module* and *transport_config* options are allowed. The order of these is significant in this case (and only in this case), a *transport_module* being paired with the first *transport_config* following it in the options list, or the default value for trailing modules. Transport starts will be attempted with each of the modules in order until one establishes a connection within the corresponding timeout (see below) or all fail.

```
{watchdog_config, [{okay|suspect, non_neg_integer()}]}
```

Configuration that alters the behaviour of the watchdog state machine. On key *okay*, the non-negative number of answered DWR messages before transitioning from REOPEN to OKAY. On key *suspect*, the number of watchdog timeouts before transitioning from OKAY to SUSPECT when DWR is unanswered, or 0 to not make the transition.

Defaults to `[{okay, 3}, {suspect, 1}]`. Not specifying a key is equivalent to specifying the default value for that key.

Warning:

The default value is as required by RFC 3539: changing it results in non-standard behaviour that should only be used to simulate misbehaving nodes during test.

```
{watchdog_timer, TwInit}
```

```
TwInit = Unsigned32()
        | {M,F,A}
```

The RFC 3539 watchdog timer. An integer value is interpreted as the RFC's TwInit in milliseconds, a jitter of ± 2 seconds being added at each rearming of the timer to compute the RFC's Tw. An MFA is expected to return the RFC's Tw directly, with jitter applied, allowing the jitter calculation to be performed by the callback.

An integer value must be at least 6000 as required by RFC 3539. Defaults to 30000.

Unrecognized options are silently ignored but are returned unmodified by *service_info/2* and can be referred to in predicate functions passed to *remove_transport/2*.

```
transport_ref() = reference()
```

Reference returned by *add_transport/2* that identifies the configuration.

Exports

```
add_transport(SvcName, {connect|listen, [Opt]}) -> {ok, Ref} | {error, Reason}
```

Types:

```
SvcName = service_name()
Opt = transport_opt()
Ref = transport_ref()
Reason = term()
```

Add transport capability to a service.

The service will start transport processes as required in order to establish a connection with the peer, either by connecting to the peer (*connect*) or by accepting incoming connection requests (*listen*). A connecting transport establishes transport connections with at most one peer, an listening transport potentially with many.

The diameter application takes responsibility for exchanging CER/CEA with the peer. Upon successful completion of capabilities exchange the service calls each relevant application module's *peer_up/3* callback after which the caller can exchange Diameter messages with the peer over the transport. In addition to CER/CEA, the service takes responsibility for the handling of DWR/DWA and required by RFC 3539, as well as for DPR/DPA.

The returned reference uniquely identifies the transport within the scope of the service. Note that the function returns before a transport connection has been established.

Note:

It is not an error to add a transport to a service that has not yet been configured: a service can be started after configuring its transports.

```
call(SvcName, App, Request, [Opt]) -> Answer | ok | {error, Reason}
```

Types:

```
SvcName = service_name()
App = application_alias()
Request = diameter_codec:message()
Answer = term()
Opt = call_opt()
```

Send a Diameter request message.

App specifies the Diameter application in which the request is defined and callbacks to the corresponding callback module will follow as described below and in *diameter_app(3)*. Unless the *detach* option is specified, the call returns either when an answer message is received from the peer or an error occurs. In the answer case, the return value is as returned by a *handle_answer/4* callback. In the error case, whether or not the error is returned directly by diameter or from a *handle_error/4* callback depends on whether or not the outgoing request is successfully encoded for transmission to the peer, the cases being documented below.

If there are no suitable peers, or if *pick_peer/4* rejects them by returning `false`, then `{error, no_connection}` is returned. Otherwise *pick_peer/4* is followed by a *prepare_request/3* callback, the message is encoded and then sent.

There are several error cases which may prevent an answer from being received and passed to a *handle_answer/4* callback:

- If the initial encode of the outgoing request fails, then the request process fails and `{error, encode}` is returned.
- If the request is successfully encoded and sent but the answer times out then a *handle_error/4* callback takes place with `Reason = timeout`.
- If the request is successfully encoded and sent but the service in question is stopped before an answer is received then a *handle_error/4* callback takes place with `Reason = cancel`.
- If the transport connection with the peer goes down after the request has been sent but before an answer has been received then an attempt is made to resend the request to an alternate peer. If no such peer is available, or if the subsequent *pick_peer/4* callback rejects the candidates, then a *handle_error/4* callback takes place with `Reason = failover`. If a peer is selected then a *prepare_retransmit/3* callback takes place, after which the semantics are the same as following an initial *prepare_request/3* callback.
- If an encode error takes place during retransmission then the request process fails and `{error, failure}` is returned.
- If an application callback made in processing the request fails (*pick_peer*, *prepare_request*, *prepare_retransmit*, *handle_answer* or *handle_error*) then either `{error, encode}` or `{error, failure}` is returned depending on whether or not there has been an attempt to send the request over the transport.

Note that `{error, encode}` is the only return value which guarantees that the request has **not** been sent over the transport connection.

origin_state_id() -> Unsigned32()

Return a reasonable value for use as Origin-State-Id in outgoing messages.

The value returned is the number of seconds since 19680120T031408Z, the first value that can be encoded as a Diameter *Time()*, at the time the diameter application was started.

remove_transport(SvcName, Pred) -> ok | {error, Reason}

Types:

```
SvcName = service_name()
Pred = Fun | MFA | transport_ref() | list() | true | false
Fun = fun((transport_ref(), connect|listen, list()) -> boolean())
      | fun((transport_ref(), list()) -> boolean())
      | fun((list()) -> boolean())
MFA = {atom(), atom(), list()}
Reason = term()
```

Remove previously added transports.

Pred determines which transports to remove. An arity-3-valued *Pred* removes all transports for which *Pred*(*Ref*, *Type*, *Opts*) returns `true`, where *Type* and *Opts* are as passed to *add_transport/2* and *Ref* is as returned by it. The remaining forms are equivalent to an arity-3 fun as follows.

```
Pred = fun(transport_ref(), list()): fun(Ref, _, Opts) -> Pred(Ref, Opts) end
Pred = fun(list()): fun(_, _, Opts) -> Pred(Opts) end
Pred = transport_ref(): fun(Ref, _, _) -> Pred == Ref end
```

```

Pred = list():          fun(_, _, Opts) -> [] == Pred -- Opts end
Pred = true:           fun(_, _, _) -> true end
Pred = false:          fun(_, _, _) -> false end
Pred = {M,F,A}:        fun(Ref, Type, Opts) -> apply(M, F, [Ref, Type, Opts | A]) end

```

Removing a transport causes the corresponding transport processes to be terminated. Whether or not a DPR message is sent to a peer is controlled by value of *disconnect_cb* configured on the transport.

service_info(SvcName, Info) -> term()

Types:

SvcName = service_name()

Info = Item | [Info]

Item = atom()

Return information about a started service. Requesting info for an unknown service causes `undefined` to be returned. Requesting a list of items causes a tagged list to be returned.

Item can be one of the following.

```

'Origin-Host'
'Origin-Realm'
'Vendor-Id'
'Product-Name'
'Origin-State-Id'
'Host-IP-Address'
'Supported-Vendor'
'Auth-Application-Id'
'Inband-Security-Id'
'Acct-Application-Id'
'Vendor-Specific-Application-Id'
'Firmware-Revision'

```

Return a capability value as configured with *start_service/2*.

applications

Return the list of applications as configured with *start_service/2*.

capabilities

Return a tagged list of all capabilities values as configured with *start_service/2*.

transport

Return a list containing one entry for each of the service's transport as configured with *add_transport/2*. Each entry is a tagged list containing both configuration and information about established peer connections. An example return value with for a client service with Origin-Host "client.example.com" configured with a single transport connected to "server.example.com" might look as follows.

```

[[ {ref, #Ref<0.0.0.93>,
  {type, connect},
  {options, [ {transport_module, diameter_tcp},
              {transport_config, [ {ip, {127,0,0,1}},
                                   {raddr, {127,0,0,1}},
                                   {rport, 3868},
                                   {reuseaddr, true} ] ] }},
  {watchdog, {<0.66.0>, {1346,171491,996448}, okay}},
  {peer, {<0.67.0>, {1346,171491,999906}}},

```

```

{apps, [{0, common}]},
{caps, [{origin_host, {"client.example.com", "server.example.com"}},
        {origin_realm, {"example.com", "example.com"}},
        {host_ip_address, [{127, 0, 0, 1}], [127, 0, 0, 1]}]},
{vendor_id, {0, 193}},
{product_name, {"Client", "Server"}},
{origin_state_id, {[], []}},
{supported_vendor_id, {[], []}},
{auth_application_id, {[0], [0]}},
{inband_security_id, {[], [0]}},
{acct_application_id, {[], []}},
{vendor_specific_application_id, {[], []}},
{firmware_revision, {[], []}},
{avp, {[], []}}},
{port, [{owner, <0.69.0>},
        {module, diameter_tcp},
        {socket, {{127, 0, 0, 1}, 48758}},
        {peer, {{127, 0, 0, 1}, 3868}},
        {statistics, [{recv_oct, 656},
                     {recv_cnt, 6},
                     {recv_max, 148},
                     {recv_avg, 109},
                     {recv_dvi, 19},
                     {send_oct, 836},
                     {send_cnt, 6},
                     {send_max, 184},
                     {send_avg, 139},
                     {send_pend, 0}]}]},
{statistics, [{{0, 258, 0}, recv}, 3},
              {{0, 258, 1}, send}, 3},
              {{0, 258, 0}, recv, {'Result-Code', 2001}}, 3},
              {{0, 257, 0}, recv}, 1},
              {{0, 257, 1}, send}, 1},
              {{0, 257, 0}, recv, {'Result-Code', 2001}}, 1},
              {{0, 280, 1}, recv}, 2},
              {{0, 280, 0}, send}, 2},
              {{0, 280, 0}, send, {'Result-Code', 2001}}, 2}}]}]}

```

Here `ref` is a `transport_ref()` and `options` the corresponding `transport_opt()` list passed to `add_transport/2`. The `watchdog` entry shows the state of a connection's RFC 3539 watchdog state machine. The `peer` entry identifies the `diameter_app:peer_ref()` for which there will have been `peer_up/3` callbacks for the Diameter applications identified by the `apps` entry, `common` being the `application_alias()`. The `caps` entry identifies the capabilities sent by the local node and received from the peer during capabilities exchange. The `port` entry displays socket-level information about the transport connection. The `statistics` entry presents Diameter-level counters, an entry like `{{0, 280, 1}, recv}, 2` saying that the client has received 2 DWR messages: `{0, 280, 1} = {Application_Id, Command_Code, R_Flag}`.

Note that `watchdog`, `peer`, `apps`, `caps` and `port` entries depend on connectivity with the peer and may not be present. Note also that the `statistics` entry presents values accumulated during the lifetime of the transport configuration.

A listening transport presents its information slightly differently since there may be multiple accepted connections for the same `transport_ref()`. The transport info returned by a server with a single client connection might look as follows.

```

[{{ref, #Ref<0.0.0.61>},
 {type, listen},
 {options, [{transport_module, diameter_tcp},
            {transport_config, [{reuseaddr, true},
                               {ip, {127, 0, 0, 1}},
                               ]}]}]}]}

```

```

        {port,3868}]}}},
{accept,[{watchdog,<0.56.0>,{1346,171481,226895},okay}},
        {peer,<0.58.0>,{1346,171491,999511}}},
        {apps,[{0,common}}},
        {caps,[{origin_host,{"server.example.com","client.example.com"}},
              {origin_realm,{"example.com","example.com"}},
              {host_ip_address,[{127,0,0,1},{127,0,0,1}}},
              {vendor_id,{193,0}},
              {product_name,{"Server","Client"}},
              {origin_state_id,[[],[]}},
              {supported_vendor_id,[[],[]}},
              {auth_application_id,[{0},{0}}},
              {inband_security_id,[[],[]}},
              {acct_application_id,[[],[]}},
              {vendor_specific_application_id,[[],[]}},
              {firmware_revision,[[],[]}},
              {avp,[[],[]}}}],
        {port,[{owner,<0.62.0>,
              {module,diameter_tcp},
              {socket,[{127,0,0,1},3868}},
              {peer,[{127,0,0,1},48758}},
              {statistics,[{recv_oct,1576},
                          {recv_cnt,16},
                          {recv_max,184},
                          {recv_avg,98},
                          {recv_dvi,26},
                          {send_oct,1396},
                          {send_cnt,16},
                          {send_max,148},
                          {send_avg,87},
                          {send_pend,0}]}]}],
        [{watchdog,<0.72.0>,{1346,171491,998404},initial}]}]}],
{statistics,[{{0,280,0},recv},7},
            {{0,280,1},send},7},
            {{0,280,0},recv,{'Result-Code',2001}},7},
            {{0,258,1},recv},3},
            {{0,258,0},send},3},
            {{0,258,0},send,{'Result-Code',2001}},3},
            {{0,280,1},recv},5},
            {{0,280,0},send},5},
            {{0,280,0},send,{'Result-Code',2001}},5},
            {{0,257,1},recv},1},
            {{0,257,0},send},1},
            {{0,257,0},send,{'Result-Code',2001}},1}]}]}]

```

The information presented here is as in the `connect` case except that the client connections are grouped under an `accept` tuple.

Whether or not the `transport_opt()` `pool_size` has been configured affects the format of the listing in the case of a connecting transport, since a value greater than 1 implies multiple transport processes for the same `transport_ref()`, as in the listening case. The format in this case is similar to the listening case, with a pool tuple in place of an `accept` tuple.

connections

Return a list containing one entry for every established transport connection whose watchdog state machine is not in the down state. This is a flat view of `transport` info which lists only active connections and for which Diameter-level statistics are accumulated only for the lifetime of the transport connection. A return value for the server above might look as follows.

```
[[{ref,#Ref<0.0.0.61>},
```

```

{type,accept},
{options,[{transport_module,diameter_tcp},
          {transport_config,[{reuseaddr,true},
                             {ip,{127,0,0,1}},
                             {port,3868}]}]},
{watchdog,{<0.56.0>,{1346,171481,226895},okay}},
{peer,{<0.58.0>,{1346,171491,999511}}},
{apps,[{0,common}]},
{caps,[{origin_host,{"server.example.com","client.example.com"}},
        {origin_realm,{"example.com","example.com"}},
        {host_ip_address,[{127,0,0,1},{127,0,0,1}]},
        {vendor_id,{193,0}},
        {product_name,{"Server","Client"}},
        {origin_state_id,[[],[]]},
        {supported_vendor_id,[[],[]]},
        {auth_application_id,[{0},{0}]},
        {inband_security_id,[[],[]]},
        {acct_application_id,[[],[]]},
        {vendor_specific_application_id,[[],[]]},
        {firmware_revision,[[],[]]},
        {avp,[[],[]]}]},
{port,[{owner,<0.62.0>},
        {module,diameter_tcp},
        {socket,[{127,0,0,1},3868]},
        {peer,[{127,0,0,1},48758]},
        {statistics,[{recv_oct,10124},
                    {recv_cnt,132},
                    {recv_max,184},
                    {recv_avg,76},
                    {recv_dvi,9},
                    {send_oct,10016},
                    {send_cnt,132},
                    {send_max,148},
                    {send_avg,75},
                    {send_pend,0}]}]},
{statistics,[{{0,280,0},recv},62},
             {{0,280,1},send},62},
             {{0,280,0},recv,{'Result-Code',2001}},62},
             {{0,258,1},recv},3},
             {{0,258,0},send},3},
             {{0,258,0},send,{'Result-Code',2001}},3},
             {{0,280,1},recv},66},
             {{0,280,0},send},66},
             {{0,280,0},send,{'Result-Code',2001}},66},
             {{0,257,1},recv},1},
             {{0,257,0},send},1},
             {{0,257,0},send,{'Result-Code',2001}},1}]}]}

```

Note that there may be multiple entries with the same ref, in contrast to transport info.

statistics

Return a `{{Counter, Ref}, non_neg_integer()}` list of counter values. Ref can be either a `transport_ref()` or a `diameter_app:peer_ref()`. Entries for the latter are folded into corresponding entries for the former as peer connections go down. Entries for both are removed at `remove_transport/2`. The Diameter-level statistics returned by `transport` and `connections` info are based upon these entries.

diameter_app:peer_ref()

Return transport configuration associated with a single peer, as passed to `add_transport/2`. The returned list is empty if the peer is unknown. Otherwise it contains the ref, type and options tuples as in `transport` and `connections` info above. For example:

```
[{ref,#Ref<0.0.0.61>},
 {type,accept},
 {options,[{transport_module,diameter_tcp},
           {transport_config,[{reuseaddr,true},
                              {ip,{127,0,0,1}},
                              {port,3868}]}]}]}
```

services() -> [SvcName]

Types:

SvcName = *service_name()*

Return the list of started services.

session_id(Ident) -> OctetString()

Types:

Ident = *DiameterIdentity()*

Return a value for a Session-Id AVP.

The value has the form required by section 8.8 of RFC 6733. Ident should be the Origin-Host of the peer from which the message containing the returned value will be sent.

start() -> ok | {error, Reason}

Start the diameter application.

The diameter application must be started before starting a service. In a production system this is typically accomplished by a boot file, not by calling `start/0` explicitly.

start_service(SvcName, Options) -> ok | {error, Reason}

Types:

SvcName = *service_name()*

Options = [*service_opt()*]

Reason = *term()*

Start a diameter service.

A service defines a locally-implemented Diameter node, specifying the capabilities to be advertised during capabilities exchange. Transports are added to a service using *add_transport/2*.

Note:

A transport can both override its service's capabilities and restrict its supported Diameter applications so "service = Diameter node as identified by Origin-Host" is not necessarily the case.

stop() -> ok | {error, Reason}

Stop the diameter application.

stop_service(SvcName) -> ok | {error, Reason}

Types:

diameter

```
SvcName = service_name()  
Reason = term()
```

Stop a diameter service.

Stopping a service causes all associated transport connections to be broken. A DPR message will be sent as in the case of *remove_transport/2*.

Note:

Stopping a service does not remove any associated transports: *remove_transport/2* must be called to remove transport configuration.

```
subscribe(SvcName) -> true
```

Types:

```
SvcName = service_name()
```

Subscribe to *service_event()* messages from a service.

It is not an error to subscribe to events from a service that does not yet exist. Doing so before adding transports is required to guarantee the reception of all transport-related events.

```
unsubscribe(SvcName) -> true
```

Types:

```
SvcName = service_name()
```

Unsubscribe to event messages from a service.

SEE ALSO

diameter_app(3), *diameter_transport(3)*, *diameter_dict(4)*

diameterc

Command

The `diameterc` utility is used to compile a diameter *dictionary file* into Erlang source. The resulting source implements the interface `diameter` required to encode and decode the dictionary's messages and AVPs.

The module `diameter_make(3)` provides an alternate compilation interface.

USAGE

`diameterc` [<options>] <file>

Compile a single dictionary file to Erlang source. Valid options are as follows.

`-i <dir>`

Prepend the specified directory to the code path. Use to point at beam files compiled from inherited dictionaries, `@inherits` in a dictionary file creating a beam dependency, not an erl/hrl dependency.

Multiple `-i` options can be specified.

`-o <dir>`

Write generated source to the specified directory. Defaults to the current working directory.

`-E`

`-H`

Suppress erl and hrl generation, respectively.

`--name <name>`

`--prefix <prefix>`

Transform the input dictionary before compilation, setting `@name` or `@prefix` to the specified string.

`--inherits <arg>`

Transform the input dictionary before compilation, appending `@inherits` of the specified string.

Two forms of `--inherits` have special meaning:

```
--inherits -  
--inherits Prev/Mod
```

The first has the effect of clearing any previous inherits, the second of replacing a previous inherits of `Prev` to one of `Mod`. This allows the semantics of the input dictionary to be changed without modifying the file itself.

Multiple `--inherits` options can be specified.

EXIT STATUS

Returns 0 on success, non-zero on failure.

SEE ALSO

`diameter_make(3)`, `diameter_dict(4)`

diameter_app

Erlang module

A diameter service as started by *diameter:start_service/2* configures one or more Diameter applications, each of whose configuration specifies a callback that handles messages specific to the application. The messages and AVPs of the application are defined in a dictionary file whose format is documented in *diameter_dict(4)* while the callback module is documented here. The callback module implements the Diameter application-specific functionality of a service.

A callback module must export all of the functions documented below. The functions themselves are of three distinct flavours:

- *peer_up/3* and *peer_down/3* signal the attainment or loss of connectivity with a Diameter peer.
- *pick_peer/4*, *prepare_request/3*, *prepare_retransmit/3*, *handle_answer/4* and *handle_error/4* are (or may be) called as a consequence of a call to *diameter:call/4* to send an outgoing Diameter request message.
- *handle_request/3* is called in response to an incoming Diameter request message.

The arities for the the callback functions here assume no extra arguments. All functions will also be passed any extra arguments configured with the callback module itself when calling *diameter:start_service/2* and, for the call-specific callbacks, any extra arguments passed to *diameter:call/4*.

DATA TYPES

`capabilities()` = `#diameter_caps{}`

A record containing the identities of the local Diameter node and the remote Diameter peer having an established transport connection, as well as the capabilities as determined by capabilities exchange. Each field of the record is a 2-tuple consisting of values for the (local) host and (remote) peer. Optional or possibly multiple values are encoded as lists of values, mandatory values as the bare value.

`message()` = `diameter_codec:message()`

The representation of a Diameter message as passed to *diameter:call/4* or returned from a *handle_request/3* callback.

`packet()` = `diameter_codec:packet()`

A container for incoming and outgoing Diameter messages that's passed through encode/decode and transport. Fields should not be set in return values except as documented.

`peer_ref()` = `term()`

A term identifying a transport connection with a Diameter peer.

`peer()` = `{peer_ref(), capabilities()}`

A tuple representing a Diameter peer connection.

`state()` = `term()`

The state maintained by the application callback functions *peer_up/3*, *peer_down/3* and (optionally) *pick_peer/4*. The initial state is configured in the call to *diameter:start_service/2* that configures the application on a service. Callback functions returning a state are evaluated in a common service-specific process while those not returning state are evaluated in a request-specific process.

Exports

Mod:peer_up(SvcName, Peer, State) -> NewState

Types:

```

SvcName = diameter:service_name()
Peer = peer()
State = NewState = state()

```

Invoked to signal the availability of a peer connection on the local Erlang node. In particular, capabilities exchange with the peer has indicated support for the application in question, the RFC 3539 watchdog state machine for the connection has reached state OKAY and Diameter messages can be both sent and received.

Note:

A watchdog state machine can reach state OKAY from state SUSPECT without a new capabilities exchange taking place. A new transport connection (and capabilities exchange) results in a new `peer_ref()`.

Note:

There is no requirement that a callback return before incoming requests are received: `handle_request/3` callbacks must be handled independently of `peer_up/3` and `peer_down/3`.

Mod:peer_down(SvcName, Peer, State) -> NewState

Types:

```

SvcName = diameter:service_name()
Peer = peer()
State = NewState = state()

```

Invoked to signal that a peer connection on the local Erlang node is no longer available following a previous call to `peer_up/3`. In particular, that the RFC 3539 watchdog state machine for the connection has left state OKAY and the peer will no longer be a candidate in `pick_peer/4` callbacks.

Mod:pick_peer(LocalCandidates, RemoteCandidates, SvcName, State) -> Selection | false

Types:

```

LocalCandidates = RemoteCandidates = [peer()]
SvcName = diameter:service_name()
State = NewState = state()
Selection = {ok, Peer} | {Peer, NewState}
Peer = peer() | false

```

Invoked as a consequence of a call to `diameter:call/4` to select a destination peer for an outgoing request. The return value indicates the selected peer.

The candidate lists contain only those peers that have advertised support for the Diameter application in question during capabilities exchange, that have not be excluded by a `filter` option in the call to `diameter:call/4` and whose

watchdog state machine is in the OKAY state. The order of the elements is unspecified except that any peers whose Origin-Host and Origin-Realm matches that of the outgoing request (in the sense of a `{filter, {all, [host, realm]}}` option to `diameter:call/4`) will be placed at the head of the list. `LocalCandidates` contains peers whose transport process resides on the local Erlang node while `RemoteCandidates` contains peers that have been communicated from other nodes by services of the same name.

A callback that returns a `peer()` will be followed by a `prepare_request/3` callback and, if the latter indicates that the request should be sent, by either `handle_answer/4` or `handle_error/4` depending on whether or not an answer message is received from the peer. If the transport becomes unavailable after `prepare_request/3` then a new `pick_peer/4` callback may take place to failover to an alternate peer, after which `prepare_retransmit/3` takes the place of `prepare_request/3` in resending the request. There is no guarantee that a `pick_peer/4` callback to select an alternate peer will be followed by any additional callbacks since a retransmission to an alternate peer is abandoned if an answer is received from a previously selected peer.

The return values `false` and `{false, State}` (that is, `NewState = State`) are equivalent, as are `{ok, Peer}` and `{Peer, State}`.

Note:

The `diameter:service_opt()` `use_shared_peers` determines whether or not a service uses peers shared from other nodes. If not then `RemoteCandidates` is the empty list.

Warning:

The return value `{Peer, NewState}` is only allowed if the Diameter application in question was configured with the `diameter:application_opt()` `{call_mutates_state, true}`. Otherwise, the `State` argument is always the initial value as configured on the application, not any subsequent value returned by a `peer_up/3` or `peer_down/3` callback.

Mod:prepare_request(Packet, SvcName, Peer) -> Action

Types:

```
Packet = packet()
SvcName = diameter:service_name()
Peer = peer()
Action = Send | Discard | {eval_packet, Action, PostF}
Send = {send, packet() | message()}
Discard = {discard, Reason} | discard
PostF = diameter:evaluable()
```

Invoked to return a request for encoding and transport. Allows the sender to use the selected peer's capabilities to modify the outgoing request. Many implementations may simply want to return `{send, Packet}`

A returned `packet()` should set the request to be encoded in its `msg` field and can set the `transport_data` field in order to pass information to the transport process. Extra arguments passed to `diameter:call/4` can be used to communicate transport (or any other) data to the callback.

A returned `packet()` can set the `header` field to a `#diameter_header{}` to specify values that should be preserved in the outgoing request, values otherwise being those in the header record contained in `Packet`. A returned `length`, `cmd_code` or `application_id` is ignored.

A returned `PostF` will be evaluated on any encoded `#diameter_packet{}` prior to transmission, the `bin` field containing the encoded binary. The return value is ignored.

Returning `{discard, Reason}` causes the request to be aborted and the `diameter:call/4` for which the callback has taken place to return `{error, Reason}`. Returning `discard` is equivalent to returning `{discard, discarded}`.

Mod:prepare_retransmit(Packet, SvcName, Peer) -> Action

Types:

```

Packet = packet()
SvcName = diameter:service_name()
Peer = peer()
Action = Send | Discard | {eval_packet, Action, PostF}
Send = {send, packet() | message()}
Discard = {discard, Reason} | discard
PostF = diameter:evaluable()

```

Invoked to return a request for encoding and retransmission. Has the same role as `prepare_request/3` in the case that a peer connection is lost and an alternate peer selected but the argument `packet()` is as returned by the initial `prepare_request/3`.

Returning `{discard, Reason}` causes the request to be aborted and a `handle_error/4` callback to take place with `Reason` as initial argument. Returning `discard` is equivalent to returning `{discard, discarded}`.

Mod:handle_answer(Packet, Request, SvcName, Peer) -> Result

Types:

```

Packet = packet()
Request = message()
SvcName = diameter:service_name()
Peer = peer()
Result = term()

```

Invoked when an answer message is received from a peer. The return value is returned from `diameter:call/4` unless the `detach` option was specified.

The decoded answer record and undecoded binary are in the `msg` and `bin` fields of the argument `packet()` respectively. `Request` is the outgoing request message as was returned from `prepare_request/3` or `prepare_retransmit/3`.

For any given call to `diameter:call/4` there is at most one `handle_answer/4` callback: any duplicate answer (due to retransmission or otherwise) is discarded. Similarly, only one of `handle_answer/4` or `handle_error/4` is called.

By default, an incoming answer message that cannot be successfully decoded causes the request process to fail, causing `diameter:call/4` to return `{error, failure}` unless the `detach` option was specified. In particular, there is no `handle_error/4` callback in this case. The `diameter:application_opt()` `answer_errors` can be set to change this behaviour.

Mod:handle_error(Reason, Request, SvcName, Peer) -> Result

Types:

```

Reason = timeout | failover | term()
Request = message()
SvcName = diameter:service_name()

```

```
Peer = peer()
Result = term()
```

Invoked when an error occurs before an answer message is received in response to an outgoing request. The return value is returned from *diameter:call/4* unless the *detach* option was specified.

Reason *timeout* indicates that an answer message has not been received within the time specified with the corresponding *diameter:call_opt()*. Reason *failover* indicates that the transport connection to the peer to which the request has been sent has become unavailable and that no alternate peer was not selected.

Mod:handle_request(Packet, SvcName, Peer) -> Action

Types:

```
Packet = packet()
SvcName = term()
Peer = peer()
Action = Reply | {relay, [Opt]} | discard | {eval|eval_packet, Action, PostF}
Reply = {reply, packet() | message()} | {answer_message, 3000..3999 | 5000..5999} | {protocol_error, 3000..3999}
Opt = diameter:call_opt()
PostF = diameter:evaluable()
```

Invoked when a request message is received from a peer. The application in which the callback takes place (that is, the callback module as configured with *diameter:start_service/2*) is determined by the Application Identifier in the header of the incoming request message, the selected module being the one whose corresponding dictionary declares itself as defining either the application in question or the Relay application.

The argument *packet()* has the following signature.

```
#diameter_packet{header = #diameter_header{},
  avps = [#diameter_avp{}],
  msg = record() | undefined,
  errors = [Unsigned32() | {Unsigned32(), #diameter_avp{}]},
  bin = binary(),
  transport_data = term()}
```

The *msg* field will be *undefined* in case the request has been received in the relay application. Otherwise it contains the record representing the request as outlined in *diameter_dict(4)*.

The *errors* field specifies any results codes identifying errors found while decoding the request. This is used to set Result-Code and/or Failed-AVP in a returned answer unless the callback returns a *#diameter_packet{}* whose *errors* field is set to either a non-empty list of its own, in which case this list is used instead, or the atom *false* to disable any setting of Result-Code and Failed-AVP. Note that the errors detected by diameter are of the 3xxx and 5xxx series, Protocol Errors and Permanent Failures respectively. The *errors* list is empty if the request has been received in the relay application.

The *transport_data* field contains an arbitrary term passed into diameter from the transport module in question, or the atom *undefined* if the transport specified no data. The term is preserved if a *message()* is returned but must be set explicitly in a returned *packet()*.

The semantics of each of the possible return values are as follows.

```
{reply, packet() | message()}
```

Send the specified answer message to the peer. In the case of a *packet()*, the message to be sent must be set in the *msg* field and the *header* field can be set to a `#diameter_header{ }` to specify values that should be preserved in the outgoing answer, appropriate values otherwise being set by diameter.

```
{answer_message, 3000..3999 | 5000..5999}
```

Send an answer message to the peer containing the specified Result-Code. Equivalent to

```
{reply, ['answer-message' | Avps]}
```

where *Avps* sets the Origin-Host, Origin-Realm, the specified Result-Code and (if the request contained one) Session-Id AVPs, and possibly Failed-AVP as described below.

Returning a value other than 3xxx or 5xxx will cause the request process in question to fail, as will returning a 5xxx value if the peer connection in question has been configured with the RFC 3588 common dictionary `diameter_gen_base_rfc3588`. (Since RFC 3588 only allows 3xxx values in an answer-message.)

When returning 5xxx, Failed-AVP will be populated with the AVP of the first matching Result-Code/AVP pair in the *errors* field of the argument *packet()*, if found. If this is not appropriate then an answer-message should be constructed explicitly and returned in a *reply* tuple instead.

```
{relay, Opts}
```

Relay a request to another peer in the role of a Diameter relay agent. If a routing loop is detected then the request is answered with 3005 (DIAMETER_LOOP_DETECTED). Otherwise a Route-Record AVP (containing the sending peer's Origin-Host) is added to the request and *pick_peer/4* and subsequent callbacks take place just as if *diameter:call/4* had been called explicitly. The End-to-End Identifier of the incoming request is preserved in the header of the relayed request.

The returned *Opts* should not specify *detach*. A subsequent *handle_answer/4* callback for the relayed request must return its first argument, the *packet()* containing the answer message. Note that the *extra* option can be specified to supply arguments that can distinguish the relay case from others if so desired. Any other return value (for example, from a *handle_error/4* callback) causes the request to be answered with 3002 (DIAMETER_UNABLE_TO_DELIVER).

```
discard
```

Discard the request. No answer message is sent to the peer.

```
{eval, Action, PostF}
```

Handle the request as if *Action* has been returned and then evaluate *PostF* in the request process. The return value is ignored.

```
{eval_packet, Action, PostF}
```

Like *eval* but evaluate *PostF* on any encoded `#diameter_packet{ }` prior to transmission, the *bin* field containing the encoded binary. The return value is ignored.

```
{protocol_error, 3000..3999}
```

Equivalent to `{answer_message, 3000..3999}`.

Note:

Requests containing errors may be answered by diameter, without a callback taking place, depending on the value of the *diameter:application_opt()* request_errors.

diameter_codec

Erlang module

Incoming Diameter messages are decoded from `binary()` before being communicated to `diameter_app(3)` callbacks. Similarly, outgoing Diameter messages are encoded into `binary()` before being passed to the appropriate `diameter_transport(3)` module for transmission. The functions in this module implement this encode/decode.

Note:

Calls to this module are made by diameter itself as a consequence of configuration passed to `diameter:start_service/2`. The encode/decode functions may also be useful for other purposes (eg. test) but the diameter user does not need to call them explicitly when sending and receiving messages using `diameter:call/4` and the callback interface documented in `diameter_app(3)`.

The `header()` and `packet()` records below are defined in `diameter.hrl`, which can be included as follows.

```
-include_lib("diameter/include/diameter.hrl").
```

Application-specific records are defined in the hrl files resulting from dictionary file compilation.

DATA TYPES

```
uint8() = 0..255
uint24() = 0..16777215
uint32() = 0..4294967295
```

8-bit, 24-bit and 32-bit integers occurring in Diameter and AVP headers.

```
avp() = #diameter_avp{}
```

The application-neutral representation of an AVP. Primarily intended for use by relay applications that need to handle arbitrary Diameter applications. A service implementing a specific Diameter application (for which it configures a dictionary) can manipulate values of type `message()` instead.

Fields have the following types.

```
code = uint32()
is_mandatory = boolean()
need_encryption = boolean()
vendor_id = uint32() | undefined
```

Values in the AVP header, corresponding to AVP Code, the M flag, P flags and Vendor-ID respectively. A Vendor-ID other than `undefined` implies a set V flag.

```
data = iolist()
```

The data bytes of the AVP.

```
name = atom()
```

The name of the AVP as defined in the dictionary file in question, or `undefined` if the AVP is unknown to the dictionary file in question.

`value = term()`

The decoded value of an AVP. Will be undefined on decode if the data bytes could not be decoded or the AVP is unknown. The type of a decoded value is as document in *diameter_dict(4)*.

`type = atom()`

The type of the AVP as specified in the dictionary file in question (or one it inherits). Possible types are undefined and the Diameter types: `OctetString`, `Integer32`, `Integer64`, `Unsigned32`, `Unsigned64`, `Float32`, `Float64`, `Grouped`, `Enumerated`, `Address`, `Time`, `UTF8String`, `DiameterIdentity`, `DiameterURI`, `IPFilterRule` and `QoSFilterRule`.

`dictionary() = module()`

The name of a generated dictionary module as generated by *diameterc(1)* or *diameter_make:codec/2*. The interface provided by a dictionary module is an implementation detail that may change.

`header() = #diameter_header{}`

The record representation of the Diameter header. Values in a *packet()* returned by *decode/2* are as extracted from the incoming message. Values set in an *packet()* passed to *encode/2* are preserved in the encoded binary(), with the exception of `length`, `cmd_code` and `application_id`, all of which are determined by the *dictionary()* in question.

Note:

It is not necessary to set header fields explicitly in outgoing messages as diameter itself will set appropriate values. Setting inappropriate values can be useful for test purposes.

Fields have the following types.

```
version = uint8()
length = uint24()
cmd_code = uint24()
application_id = uint32()
hop_by_hop_id = uint32()
end_to_end_id = uint32()
```

Values of the Version, Message Length, Command-Code, Application-ID, Hop-by-Hop Identifier and End-to-End Identifier fields of the Diameter header.

```
is_request = boolean()
is_proxiable = boolean()
is_error = boolean()
is_retransmitted = boolean()
```

Values corresponding to the R(equest), P(roxiable), E(rror) and T(Potentially re-transmitted message) flags of the Diameter header.

`message() = record() | list()`

The representation of a Diameter message as passed to *diameter:call/4* or returned from a *handle_request/3* callback. The record representation is as outlined in *diameter_dict(4)*: a message as defined in a dictionary file is encoded as a record with one field for each component AVP. Equivalently, a message can also be encoded as a list whose head is the atom-valued message name (as specified in the relevant dictionary file) and whose tail is a list of {AvpName, AvpValue} pairs.

Another list-valued representation allows a message to be specified as a list whose head is a *header()* and whose tail is an *avp()* list. This representation is used by diameter itself when relaying requests as directed by the return value of a *handle_request/3* callback. It differs from the other two in that it bypasses the checks for messages that do not agree with their definitions in the dictionary in question: messages are sent exactly as specified.

```
packet() = #diameter_packet{}
```

A container for incoming and outgoing Diameter messages. Fields have the following types.

```
header = header() | undefined
```

The Diameter header of the message. Can be (and typically should be) `undefined` for an outgoing message in a non-relay application, in which case diameter provides appropriate values.

```
avps = [avp()] | undefined
```

The AVPs of the message. Ignored for an outgoing message if the `msg` field is set to a value other than `undefined`.

```
msg = message() | undefined
```

The incoming/outgoing message. For an incoming message, a record if the message can be decoded in a non-relay application, `undefined` otherwise. For an outgoing message, setting a `[header() | avp()]` list is equivalent to setting the `header` and `avps` fields to the corresponding values.

Warning:

A record-valued `msg` field does **not** imply an absence of decode errors. The `errors` field should also be examined.

```
bin = binary()
```

The incoming message prior to encode or the outgoing message after encode.

```
errors = [5000..5999 | {5000..5999, avp()}]
```

Errors detected at decode of an incoming message, as identified by a corresponding 5xxx series Result-Code (Permanent Failures). For an incoming request, these should be used to formulate an appropriate answer as documented for the *handle_request/3* callback in *diameter_app(3)*. For an incoming answer, the *diameter:application_opt()* `answer_errors` determines the behaviour.

```
transport_data = term()
```

An arbitrary term of meaning only to the transport process in question, as documented in *diameter_transport(3)*.

Exports

```
decode(Mod, Bin) -> Pkt
```

Types:

```
Mod = dictionary()
```

```
Bin = binary()
```

```
Pkt = packet()
```

Decode a Diameter message.

diameter_codec

`encode(Mod, Msg) -> Pkt`

Types:

`Mod = dictionary()`

`Msg = message() | packet()`

`Pkt = packet()`

Encode a Diameter message.

SEE ALSO

`diameterc(1)`, `diameter_app(3)`, `diameter_dict(4)`, `diameter_make(3)`

diameter_dict

Name

A diameter service, as configured with *diameter:start_service/2*, specifies one or more supported Diameter applications. Each Diameter application specifies a dictionary module that knows how to encode and decode its messages and AVPs. The dictionary module is in turn generated from a file that defines these messages and AVPs. The format of such a file is defined in *FILE FORMAT* below. Users add support for their specific applications by creating dictionary files, compiling them to Erlang modules using either *diameterc(1)* or *diameter_make(3)* and configuring the resulting dictionaries modules on a service.

Dictionary module generation also results in a hrl file that defines records for the messages and Grouped AVPs defined by the dictionary, these records being what a user of the diameter application sends and receives, modulo other possible formats as discussed in *diameter_app(3)*. These records and the underlying Erlang data types corresponding to Diameter data formats are discussed in *MESSAGE RECORDS* and *DATA TYPES* respectively. The generated hrl also contains macro definitions for the possible values of AVPs of type Enumerated.

The diameter application includes five dictionary modules corresponding to applications defined in section 2.4 of RFC 6733: *diameter_gen_base_rfc3588* and *diameter_gen_base_rfc6733* for the Diameter Common Messages application with application identifier 0, *diameter_gen_accounting* (for RFC 3588) and *diameter_gen_acct_rfc6733* for the Diameter Base Accounting application with application identifier 3 and *diameter_gen_relay* the Relay application with application identifier 0xFFFFFFFF.

The Common Message and Relay applications are the only applications that diameter itself has any specific knowledge of. The Common Message application is used for messages that diameter itself handles: CER/CEA, DWR/DWA and DPR/DPA. The Relay application is given special treatment with regard to encode/decode since the messages and AVPs it handles are not specifically defined.

FILE FORMAT

A dictionary file consists of distinct sections. Each section starts with a tag followed by zero or more arguments and ends at the the start of the next section or end of file. Tags consist of an ampersand character followed by a keyword and are separated from their arguments by whitespace. Whitespace separates individual tokens but is otherwise insignificant.

The tags, their arguments and the contents of each corresponding section are as follows. Each section can occur multiple times unless otherwise specified. The order in which sections are specified is unimportant.

@id Number

Defines the integer Number as the Diameter Application Id of the application in question. Can occur at most once and is required if the dictionary defines @messages. The section has empty content.

The Application Id is set in the Diameter Header of outgoing messages of the application, and the value in the header of an incoming message is used to identify the relevant dictionary module.

Example:

```
@id 16777231
```

@name Mod

Defines the name of the generated dictionary module. Can occur at most once and defaults to the name of the dictionary file minus any extension. The section has empty content.

Note that a dictionary module should have a unique name so as not collide with existing modules in the system.

diameter_dict

Example:

```
@name etsi_e2
```

@prefix Name

Defines Name as the prefix to be added to record and constant names (followed by a '_' character) in the generated dictionary module and hrl. Can occur at most once. The section has empty content.

A prefix is optional but can be used to disambiguate between record and constant names resulting from similarly named messages and AVPs in different Diameter applications.

Example:

```
@prefix etsi_e2
```

@vendor Number Name

Defines the integer Number as the the default Vendor-Id of AVPs for which the V flag is set. Name documents the owner of the application but is otherwise unused. Can occur at most once and is required if an AVP sets the V flag and is not otherwise assigned a Vendor-Id. The section has empty content.

Example:

```
@vendor 13019 ETSI
```

@avp_vendor_id Number

Defines the integer Number as the Vendor-Id of the AVPs listed in the section content, overriding the @vendor default. The section content consists of AVP names.

Example:

```
@avp_vendor_id 2937  
WWW-Auth  
Domain-Index  
Region-Set
```

@inherits Mod

Defines the name of a dictionary module containing AVP definitions that should be imported into the current dictionary. The section content consists of the names of those AVPs whose definitions should be imported from the dictionary, an empty list causing all to be imported. Any listed AVPs must not be defined in the current dictionary and it is an error to inherit the same AVP from more than one dictionary.

Note that an inherited AVP that sets the V flag takes its Vendor-Id from either @avp_vendor_id in the inheriting dictionary or @vendor in the inherited dictionary. In particular, @avp_vendor_id in the inherited dictionary is ignored. Inheriting from a dictionary that specifies the required @vendor is equivalent to using @avp_vendor_id with a copy of the dictionary's definitions but the former makes for easier reuse.

All dictionaries should typically inherit RFC 6733 AVPs from diameter_gen_base_rfc6733.

Example:

```
@inherits diameter_gen_base_rfc6733
```

@avp_types

Defines the name, code, type and flags of individual AVPs. The section consists of definitions of the form

```
Name Code Type Flags
```

where Code is the integer AVP code, Type identifies an AVP Data Format as defined in section *DATA TYPES* below, and Flags is a string of V, M and P characters indicating the flags to be set on an outgoing AVP or a single ' - ' (minus) character if none are to be set.

Example:

```
@avp_types
Location-Information 350 Grouped MV
Requested-Information 353 Enumerated V
```

Warning:

The P flag has been deprecated by RFC 6733.

@custom_types Mod

Specifies AVPs for which module Mod provides encode/decode functions. The section contents consists of AVP names. For each such name, Mod:Name(encode|decode, Type, Data) is expected to provide encode/decode for values of the AVP, where Name is the name of the AVP, Type is it's type as declared in the @avp_types section of the dictionary and Data is the value to encode/decode.

Example:

```
@custom_types rfc4005_avps
Framed-IP-Address
```

@codecs Mod

Like @custom_types but requires the specified module to export Mod:Type(encode|decode, Name, Data) rather than Mod:Name(encode|decode, Type, Data).

Example:

```
@codecs rfc4005_avps
Framed-IP-Address
```

@messages

Defines the messages of the application. The section content consists of definitions of the form specified in section 3.2 of RFC 6733, "Command Code Format Specification".

```
@messages

RTR ::= < Diameter Header: 287, REQ, PXY >
  < Session-Id >
  { Auth-Application-Id }
  { Auth-Session-State }
  { Origin-Host }
  { Origin-Realm }
  { Destination-Host }
  { SIP-Deregistration-Reason }
  [ Destination-Realm ]
  [ User-Name ]
  * [ SIP-AOR ]
  * [ Proxy-Info ]
  * [ Route-Record ]
  * [ AVP ]

RTA ::= < Diameter Header: 287, PXY >
  < Session-Id >
  { Auth-Application-Id }
  { Result-Code }
  { Auth-Session-State }
  { Origin-Host }
  { Origin-Realm }
  [ Authorization-Lifetime ]
  [ Auth-Grace-Period ]
  [ Redirect-Host ]
  [ Redirect-Host-Usage ]
  [ Redirect-Max-Cache-Time ]
  * [ Proxy-Info ]
  * [ Route-Record ]
  * [ AVP ]
```

@grouped

Defines the contents of the AVPs of the application having type Grouped. The section content consists of definitions of the form specified in section 4.4 of RFC 6733, "Grouped AVP Values".

Example:

```
@grouped

SIP-Deregistration-Reason ::= < AVP Header: 383 >
  { SIP-Reason-Code }
  [ SIP-Reason-Info ]
  * [ AVP ]
```

Specifying a Vendor-Id in the definition of a grouped AVP is equivalent to specifying it with @avp_vendor_id.

@enum Name

Defines values of AVP Name having type Enumerated. Section content consists of names and corresponding integer values. Integer values can be prefixed with 0x to be interpreted as hexadecimal.

Note that the AVP in question can be defined in an inherited dictionary in order to introduce additional values to an enumeration otherwise defined in another dictionary.

Example:

```
@enum SIP-Reason-Code
PERMANENT_TERMINATION    0
NEW_SIP_SERVER_ASSIGNED  1
SIP_SERVER_CHANGE        2
REMOVE_SIP_SERVER        3
```

@end

Causes parsing of the dictionary to terminate: any remaining content is ignored.

Comments can be included in a dictionary file using semicolon: characters from a semicolon to end of line are ignored.

MESSAGE RECORDS

The hrl generated from a dictionary specification defines records for the messages and grouped AVPs defined in @messages and @grouped sections. For each message or grouped AVP definition, a record is defined whose name is the message or AVP name, prefixed with any dictionary prefix defined with @prefix, and whose fields are the names of the AVPs contained in the message or grouped AVP in the order specified in the definition in question. For example, the grouped AVP

```
SIP-Deregistration-Reason ::= < AVP Header: 383 >
                             { SIP-Reason-Code }
                             [ SIP-Reason-Info ]
                             * [ AVP ]
```

will result in the following record definition given an empty prefix.

```
-record('SIP-Deregistration-Reason' { 'SIP-Reason-Code',
                                     'SIP-Reason-Info',
                                     'AVP' }).
```

The values encoded in the fields of generated records depends on the type and number of times the AVP can occur. In particular, an AVP which is specified as occurring exactly once is encoded as a value of the AVP's type while an AVP with any other specification is encoded as a list of values of the AVP's type. The AVP's type is as specified in the AVP definition, the RFC 6733 types being described below.

DATA TYPES

The data formats defined in sections 4.2 ("Basic AVP Data Formats") and 4.3 ("Derived AVP Data Formats") of RFC 6733 are encoded as values of the types defined here. Values are passed to *diameter:call/4* in a request record when sending a request, returned in a resulting answer record and passed to a *handle_request/3* callback upon reception of an incoming request.

In cases in which there is a choice between string() and binary() types for OctetString() and derived types, the representation is determined by the value of *diameter:service_opt()* *string_decode*.

Basic AVP Data Formats

```
OctetString() = string() | binary()
Integer32()   = -2147483647..2147483647
Integer64()   = -9223372036854775807..9223372036854775807
Unsigned32()  = 0..4294967295
```

diameter_dict

```
Unsigned64() = 0..18446744073709551615
Float32()   = '-infinity' | float() | infinity
Float64()   = '-infinity' | float() | infinity
Grouped()   = record()
```

On encode, an OctetString() can be specified as an iolist(), excessively large floats (in absolute value) are equivalent to infinity or '-infinity' and excessively large integers result in encode failure. The records for grouped AVPs are as discussed in the previous section.

Derived AVP Data Formats

```
Address() = OctetString()
           | tuple()
```

On encode, an OctetString() IPv4 address is parsed in the usual x.x.x.x format while an IPv6 address is parsed in any of the formats specified by section 2.2 of RFC 2373, "Text Representation of Addresses". An IPv4 tuple() has length 4 and contains values of type 0..255. An IPv6 tuple() has length 8 and contains values of type 0..65535. The tuple representation is used on decode.

```
Time() = {date(), time()}
where
  date() = {Year, Month, Day}
  time() = {Hour, Minute, Second}

  Year   = integer()
  Month  = 1..12
  Day    = 1..31
  Hour   = 0..23
  Minute = 0..59
  Second = 0..59
```

Additionally, values that can be encoded are limited by way of their encoding as four octets as required by RFC 6733 with the required extension from RFC 2030. In particular, only values between $\{\{1968, 1, 20\}, \{3, 14, 8\}\}$ and $\{\{2104, 2, 26\}, \{9, 42, 23\}\}$ (both inclusive) can be encoded.

```
UTF8String() = [integer()] | binary()
```

List elements are the UTF-8 encodings of the individual characters in the string. Invalid codepoints will result in encode/decode failure. On encode, a UTF8String() can be specified as a binary, or as a nested list of binaries and codepoints.

```
DiameterIdentity() = OctetString()
```

A value must have length at least 1.

```
DiameterURI() = OctetString()
               | #diameter_URI{type = Type,
                               fqdn = FQDN,
```

```
        port = Port,  
        transport = Transport,  
        protocol = Protocol}  
  
where  
  
    Type = aaa | aaas  
    FQDN = OctetString()  
    Port = integer()  
    Transport = sctp | tcp  
    Protocol = diameter | radius | 'tacacs+'
```

On encode, fields port, transport and protocol default to 3868, sctp and diameter respectively. The grammar of an OctetString-valued DiameterURI() is as specified in section 4.3 of RFC 6733. The record representation is used on decode.

```
Enumerated() = Integer32()
```

On encode, values can be specified using the macros defined in a dictionary's hrl file.

```
IPFilterRule() = OctetString()  
QoSFilterRule() = OctetString()
```

Values of these types are not currently parsed by diameter.

SEE ALSO

diameterc(1), *diameter(3)*, *diameter_app(3)*, *diameter_codec(3)*, *diameter_make(3)*

diameter_make

Erlang module

The function `codec/2` is used to compile a diameter *dictionary file* into Erlang source. The resulting source implements the interface diameter requires to encode and decode the dictionary's messages and AVPs.

The utility `diameterc(1)` provides an alternate compilation interface.

Exports

`codec(File :: iolist() | binary(), [Opt]) -> ok | {ok, [Out]} | {error, Reason}`

Compile a single dictionary file. The input `File` can be either a path or a literal dictionary, the occurrence of newline (ascii NL) or carriage return (ascii CR) identifying the latter. `Opt` determines the format of the results and whether they are written to file or returned, and can have the following types.

`parse` | `forms` | `erl` | `hrl`

Specifies an output format. Whether the output is returned or written to file depends on whether or not option `return` is specified. When written to file, the resulting file(s) will have extensions `.D`, `.F`, `.erl`, and `.hrl` respectively, basenames defaulting to `dictionary` if the input dictionary is literal and does not specify `@name`. When returned, results are in the order of the corresponding format options. Format options default to `erl` and `hrl` (in this order) if unspecified.

The `parse` format is an internal representation that can be passed to `flatten/1` and `format/1`, while the `forms` format can be passed to `compile:forms/2`. The `erl` and `hrl` formats are returned as iolists.

`{include, string()}`

Prepend the specified directory to the code path. Use to point at beam files compiled from inherited dictionaries, `@inherits` in a dictionary file creating a beam dependency, not an erl/hrl dependency.

Multiple `include` options can be specified.

`{outdir, string()}`

Write generated source to the specified directory. Defaults to the current working directory. Has no effect if option `return` is specified.

`return`

Return results in a `{ok, [Out]}` tuple instead of writing to file and returning `ok`.

`{name|prefix, string()}`

Transform the input dictionary before compilation, setting `@name` or `@prefix` to the specified string.

`{inherits, string()}`

Transform the input dictionary before compilation, appending `@inherits` of the specified string.

Two forms have special meaning:

```
{inherits, "-"}
{inherits, "Prev/Mod"}
```

The first has the effect of clearing any previous inherits, the second of replacing a previous inherits of `Prev` to one of `Mod`. This allows the semantics of the input dictionary to be changed without modifying the file itself.

Multiple `inherits` options can be specified.

Note that a dictionary's `@name`, together with the `outdir` option, determine the output paths when the `return` option is not specified. The `@name` of a literal input dictionary defaults to `dictionary`.

A returned error reason can be converted into a readable string using `format_error/1`.

format(Parsed) -> iolist()

Turns a parsed dictionary, as returned by `codec/2`, back into the dictionary format.

flatten(Parsed) -> term()

Reconstitute a parsed dictionary, as returned by `codec/2`, without using `@inherits`. That is, construct an equivalent dictionary in which all AVP's are defined in the dictionary itself. The return value is also a parsed dictionary.

format_error(Reason) -> string()

Turn an error reason returned by `codec/2` into a readable string.

BUGS

Unrecognized options are silently ignored.

SEE ALSO

diameterc(1), *diameter_dict(4)*

diameter_transport

Erlang module

A module specified as a `transport_module` to `diameter:add_transport/2` must implement the interface documented here. The interface consists of a function with which diameter starts a transport process and a message interface with which the transport process communicates with the process that starts it (aka its parent).

DATA TYPES

`message()` = `binary()` | `diameter_codec:packet()`

A Diameter message as passed over the transport interface.

For an inbound message from a transport process, a `diameter_codec:packet()` must contain the received message in its `bin` field. In the case of an inbound request, any value set in the `transport_data` field will be passed back to the transport module in the corresponding answer message, unless the sender supplies another value.

For an outbound message to a transport process, a `diameter_codec:packet()` has a value other than `undefined` in its `transport_data` field and has the `binary()` to send in its `bin` field.

Exports

```
Mod:start({Type, Ref}, Svc, Config) -> {ok, Pid} | {ok, Pid, LAddrs} | {error, Reason}
```

Types:

```
Type = connect | accept
Ref = diameter_transport_ref()
Svc = #diameter_service{}
Config = term()
Pid = pid()
LAddrs = [inet_ip_address()]
Reason = term()
```

Start a transport process. Called by diameter as a consequence of a call to `diameter:add_transport/2` in order to establish or accept a transport connection respectively. A transport process maintains a connection with a single remote peer.

`Type` indicates whether the transport process in question is being started for a connecting (`Type=connect`) or listening (`Type=accept`) transport. In the latter case, transport processes are started as required to accept connections from multiple peers.

`Ref` is the value that was returned from the call to `diameter:add_transport/2` that has led to starting of a transport process.

`Svc` contains capabilities passed to `diameter:start_service/2` and `diameter:add_transport/2`, values passed to the latter overriding those passed to the former.

`Config` is as passed in `transport_config` tuple in the `diameter:transport_opt()` list passed to `diameter:add_transport/2`.

The start function should use the `Host-IP-Address` list in `Svc` and/or `Config` to select and return an appropriate list of local IP addresses. In the connecting case, the local address list can instead be communicated in a `connected` message (see `MESSAGES` below) following connection establishment. In either case, the local address list is used

to populate `Host-IP-Address` AVPs in outgoing capabilities exchange messages if `Host-IP-Address` is unspecified.

A transport process must implement the message interface documented below. It should retain the `pid` of its parent, monitor the parent and terminate if it dies. It should not link to the parent. It should exit if its transport connection with its peer is lost.

MESSAGES

All messages sent over the transport interface are of the form `{diameter, term()}`.

A transport process can expect messages of the following types from its parent.

```
{diameter, {send, message()}}
```

An outbound Diameter message.

```
{diameter, {close, Pid}}
```

A request to terminate the transport process after having received DPA in response to DPR. The transport process should exit. `Pid` is the `pid()` of the parent process.

```
{diameter, {tls, Ref, Type, Bool}}
```

Indication of whether or not capabilities exchange has selected inband security using TLS. `Ref` is a `reference()` that must be included in the `{diameter, {tls, Ref}}` reply message to the transport's parent process (see below). `Type` is either `connect` or `accept` depending on whether the process has been started for a connecting or listening transport respectively. `Bool` is a `boolean()` indicating whether or not the transport connection should be upgraded to TLS.

If TLS is requested (`Bool=true`) then a connecting process should initiate a TLS handshake with the peer and an accepting process should prepare to accept a handshake. A successful handshake should be followed by a `{diameter, {tls, Ref}}` message to the parent process. A failed handshake should cause the process to exit.

This message is only sent to a transport process over whose `Inband-Security-Id` configuration has indicated support for TLS.

A transport process should send messages of the following types to its parent.

```
{diameter, {self(), connected}}
```

Inform the parent that the transport process with `Type=accept` has established a connection with the peer. Not sent if the transport process has `Type=connect`.

```
{diameter, {self(), connected, Remote}}
```

```
{diameter, {self(), connected, Remote, [LocalAddr]}}
```

Inform the parent that the transport process with `Type=connect` has established a connection with a peer. Not sent if the transport process has `Type=accept`. `Remote` is an arbitrary term that uniquely identifies the remote endpoint to which the transport has connected. A `LocalAddr` list has the same semantics as one returned from `start/3`.

```
{diameter, {recv, message()}}
```

An inbound Diameter message.

```
{diameter, {tls, Ref}}
```

Acknowledgment of a successful TLS handshake. `Ref` is the `reference()` received in the `{diameter, {tls, Ref, Type, Bool}}` message in response to which the reply is sent. A transport must exit if a handshake is not successful.

SEE ALSO

diameter_tcp(3), diameter_sctp(3)

diameter_tcp

Erlang module

This module implements diameter transport over TCP using *gen_tcp(3)*. It can be specified as the value of a *transport_module* option to *diameter:add_transport/2* and implements the behaviour documented in *diameter_transport(3)*. TLS security is supported, either as an upgrade following capabilities exchange or at connection establishment.

Note that the *ssl* application is required for TLS and must be started before configuring TLS capability on diameter transports.

Exports

```
start({Type, Ref}, Svc, [Opt]) -> {ok, Pid} | {ok, Pid, [LAddr]} | {error, Reason}
```

Types:

```
Type = connect | accept
Ref = diameter:transport_ref()
Svc = #diameter_service{}
Opt = OwnOpt | SslOpt | TcpOpt
Pid = pid()
LAddr = inet:ip_address()
Reason = term()
OwnOpt = {raddr, inet:ip_address()} | {rport, integer()} | {accept, Match}
| {port, integer()} | {fragment_timer, infinity | 0..16#FFFFFFFF}
SslOpt = {ssl_options, true | list()}
TcpOpt = term()
Match = inet:ip_address() | string() | [Match]
```

The *start* function required by *diameter_transport(3)*.

Options *raddr* and *rport* specify the remote address and port for a connecting transport and are not valid for a listening transport.

Option *accept* specifies remote addresses for a listening transport and is not valid for a connecting transport. If specified, a remote address that does not match one of the specified addresses causes the connection to be aborted. Multiple *accept* options can be specified. A string-valued *Match* that does not parse as an address is interpreted as a regular expression.

Option *ssl_options* must be specified for a transport that should support TLS: a value of *true* results in a TLS handshake immediately upon connection establishment while *list()* specifies options to be passed to *ssl:connect/2* or *ssl:ssl_accept/2* after capabilities exchange if TLS is negotiated.

Option *fragment_timer* specifies the timeout, in milliseconds, of a timer used to flush messages from the incoming byte stream even if the number of bytes indicated in the Message Length field of its Diameter Header have not yet been accumulated: such a message is received over the transport interface after two successive timeouts without the reception of additional bytes. Defaults to 1000.

Remaining options are any accepted by *ssl:connect/3* or *gen_tcp:connect/3* for a connecting transport, or *ssl:listen/2* or *gen_tcp:listen/2* for a listening transport, depending on whether or not *{ssl_options, true}* has been specified. Options *binary*, *packet* and *active* cannot be specified. Also, option *port* can be specified for a listening

transport to specify the local listening port, the default being the standardized 3868. Note that the option `ip` specifies the local address.

An `ssl_options` list must be specified if and only if the transport in question has set `Inband-Security-Id` to 1 (TLS), as specified to either `diameter:start_service/2` or `diameter:add_transport/2`, so that the transport process will receive notification of whether or not to commence with a TLS handshake following capabilities exchange. Failing to specify an options list on a TLS-capable transport for which TLS is negotiated will cause TLS handshake to fail. Failing to specify TLS capability when `ssl_options` has been specified will cause the transport process to wait for a notification that will not be forthcoming, which will eventually cause the RFC 3539 watchdog to take down the connection.

If an `ip` option is not specified then the first element of a non-empty `Host-IP-Address` list in `Svc` provides the local IP address. If neither is specified then the default address selected by `gen_tcp(3)` is used. In all cases, the selected address is either returned from `start/3` or passed in a `connected` message over the transport interface.

SEE ALSO

diameter(3), diameter_transport(3), gen_tcp(3), inet(3), ssl(3)

diameter_sctp

Erlang module

This module implements diameter transport over SCTP using *gen_sctp(3)*. It can be specified as the value of a *transport_module* option to *diameter:add_transport/2* and implements the behaviour documented in *diameter_transport(3)*.

Exports

```
start({Type, Ref}, Svc, [Opt]) -> {ok, Pid, [LAddr]} | {error, Reason}
```

Types:

```
Type = connect | accept
Ref = diameter:transport_ref()
Svc = #diameter_service{}
Opt = OwnOpt | SctpOpt
Pid = pid()
LAddr = inet:ip_address()
Reason = term()
OwnOpt = {raddr, inet:ip_address()} | {rport, integer()} | {accept, Match}
SctpOpt = term()
Match = inet:ip_address() | string() | [Match]
```

The start function required by *diameter_transport(3)*.

Options *raddr* and *rport* specify the remote address and port for a connecting transport and not valid for a listening transport: the former is required while latter defaults to 3868 if unspecified. Multiple *raddr* options can be specified, in which case the connecting transport in question attempts each in sequence until an association is established.

Option *accept* specifies remote addresses for a listening transport and is not valid for a connecting transport. If specified, a remote address that does not match one of the specified addresses causes the association to be aborted. Multiple *accept* options can be specified. A string-valued *Match* that does not parse as an address is interpreted as a regular expression.

Remaining options are any accepted by *gen_sctp:open/1*, with the exception of options *mode*, *binary*, *list*, *active* and *sctp_events*. Note that options *ip* and *port* specify the local address and port respectively.

Multiple *ip* options can be specified for a multihomed peer. If none are specified then the values of *Host-IP-Address* in the *diameter_service* record are used. (In particular, one of these must be specified.) Option *port* defaults to 3868 for a listening transport and 0 for a connecting transport.

Warning:

An insufficiently large receive buffer may result in a peer having to resend incoming messages: set the *inet(3)* option *recbuf* to increase the buffer size.

An insufficiently large send buffer may result in outgoing messages being discarded: set the *inet(3)* option *sndbuf* to increase the buffer size.

diameter_sctp

The `transport_data` field of record `diameter_packet` is used to communicate the stream on which an inbound message has been received, or on which an outbound message should be sent. The value will be of the form `{stream, Id}` for an inbound message passed to a `handle_request/3` or `handle_answer/4` callback. For an outbound message, `{ostream, Id}` in the return value of `handle_request/3` or `prepare_retransmit/3` sets the outbound stream, the stream id being interpreted modulo the number of outbound streams. Any other value, or not setting a value, causes successive such sends to cycle through all outbound streams.

SEE ALSO

`diameter(3)`, `diameter_transport(3)`, `gen_sctp(3)`, `inet(3)`