DIME
Internet-Draft
Intended status: Standards Track
Expires: November 18, 2013

A. B. Roach
Mozilla
E. McMurry
Tekelec

May 17, 2013

A Mechanism for Diameter Overload Control
draft-roach-dime-overload-ctrl-03

Abstract

When a Diameter server or agent becomes overloaded, it needs to be able to gracefully reduce its load, typically by informing clients to reduce or stop sending traffic for some period of time. Otherwise, it must continue to expend resources parsing and responding to Diameter messages.

This document proposes a concrete, application-independent mechanism to address the challenge of communicating load and overload state among Diameter peers, and specifies an algorithm for load abatement to address such overload conditions as they occur. The load abatement algorithm is extensible, allowing for future documents to define additional load abatement approaches.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 18, 2013.

Copyright Notice

Table of Contents

1. Introduction

When a Diameter [RFC6733] server or agent becomes overloaded, it needs to be able to gracefully reduce its load, typically by informing clients to reduce or stop sending traffic for some period of time. Otherwise, it must continue to expend resources parsing and responding to Diameter messages.

This document defines a mechanism for communicating the load and overload information among Diameter nodes. It also defines a base algorithm for shedding traffic under overload circumstances. The design of the mechanism described in this document allows for the definition of alternate load abatement algorithms as well.

The mechanism proposed in this document is heavily influenced by the work performed in the IETF Session Initiation Protocol (SIP) Overload Control Working Group, and draws on the conclusions reached by that working group after extensive network modeling.

The solution described in this document is intended to satisfy the requirements described in [I-D.ietf-dime-overload-reqs], with the exception of REQ 34. As discussed in that document, the intention of a Diameter overload mechanism is to handle overload of the actual message processing portions of Diameter servers. This is in contrast to congestion, which is the inability of the underlying switching and routing fabric of the network to carry the volume of traffic at the volume that IP hosts wish to send it. Handling of congestion is relegated to the underlying transport protocol (TCP or SCTP), and will not be discussed.

Philosophically, the approach in designing this mechanism is based on the prospect that building a base-level, fully compliant implementation should be a very simple and straightforward exercise. However, the protocol includes many additional features that may be implemented to allow Diameter nodes to apply increasingly sophisticated behaviors. This approach gives implementors the freedom to implement as sophisticated a scheme as they desire, while freeing them from the burden of unnecessary complexity. By doing so, the mechanism allows for the rapid development and deployment of the mechanism followed by a period of steady and gradual improvements as implementations become more capable.

1.1. Mechanism Properties

The core Diameter overload mechanism described in this document is fundamentally hop-by-hop. The rationale for using a hop-by-hop approach is the same as is described in section 5.1 of [RFC6357]. However, due to the fact that Diameter networks frequently have

traffic that is easily grouped into a few well-defined categories, we have added some concepts that allow Diameter agents to push back on subsets of traffic that correspond to certain well-defined and client-visible constructs (such as Destination-Host, Destination-Realm, and Application-ID). These constructs are termed "Scopes" in this document. A more complete discussion of Scopes is found in Section 2.

The key information transmitted between Diameter peers is the current server load (to allow for better balancing of traffic, so as to preempt overload in the first place) as well as an indication of overload state and severity (overload information). The actual load and overload information is conveyed as a new compound AVP, added to any Diameter messages that allow for extensibility. As discussed in section 3.2 of [RFC6733], all CCFs are encouraged to include AVP-level extensibility by inclusion of a "* [AVP]" construct in their syntax definition. The document author has conducted an extensive (although admittedly not exhaustive) audit of existing applications, and found none lacking this property. The inclusion of load and overload information in existing messages has the property that the frequency with which information can be exchanged increases as load on the system goes up.

For the purpose of grouping the several different parts of load information together, this mechanism makes use of a Grouped AVP, called "Load-Info". The Load-Info AVP may appear one or more times in any extensible command, with the restriction that each instance of the Load-Info AVP must contain different Scopes.

Load and overload information can be conveyed during times of inter-node quiescence through the use of DWR/DWA exchanges. These exchanges can also be used to proactively change the overload or load level of a server when no other transaction is ready to be sent. Finally, in the unlikely event that an application is defined that precludes the inclusion of new AVPs in its commands, DWR/DWA exchanges can be sent at any rate acceptable to the server in order to convey load and overload information.

> In [RFC3588], the DWR and DWA message syntax did not allow for the addition of new AVPs in the DWR and DWA messages. This oversight was fixed in [RFC6733]. To allow for transmission of load information on quiescent links, implementations of the mechanism described in this document are expected to correctly handle extension AVPs in DWR and DWA messages, even if such implementations have not otherwise been upgraded to support [RFC6733].

1.2. Overview of Operation

During the capabilities exchange phase of connection establishment, peers determine whether the connection will make use of the overload control mechanism; and, if so, which optional behaviors are to be employed.

The information sent between adjacent nodes includes two key metrics: Load (which, roughly speaking, provides a linear metric of how busy the node is), and Overload-Metric (which is input to the negotiated load abatement algorithm).

Message originators (whether originating a request or an answer) include one or more Load-Info AVPs in messages when they form them. These Load-Info AVPs reflect the originators' own load and overload state.

Because information is being used on a hop-by-hop basis, it is exchanged only between adjacent nodes. This means that any Diameter agent that forwards a message (request or answer) is required to remove any information received from the previous hop, and act upon it as necessary. Agents also add their own load and overload information (which may, at implementors' preference, take previous-hop information into account) into a new Load-Info AVP before sending the request or answer along.

> Because the mechanism requires affirmative indication of support in the capabilities exchange phase of connection establishment, load and overload information will never be sent to intermediaries that do not support the overload mechanism. Therefore, no special provisions need to be made for removal of information at such intermediaries -- it will simply not be sent to them.

Message recipients are responsible for reading and acting upon load and overload information that they receive in such messages.

1.3. Documentation Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Overload Scopes

In normal operation, a Diameter node may be overloaded for some but not all possible requests. For example, an agent that supports two realms (realm A and realm B in this example) may route traffic to one

set of servers for realm A, and another set of servers for realm B. If the realm A servers are overloaded but realm B servers are not, then the agent is effectively overloaded for realm A but not for realm B.

Despite the fact that Diameter agents can report on scopes that semantically map to constructs elsewhere in the network, it is important to keep in mind that overload state is still reported on a hop-by-hop basis. In other words, the overload state reported for realm A in the example above represents the aggregate of the agent's overload state along with the overload state being reported by applicable upstream servers (those serving realm A).

Even without the use of Diameter agents, similar situations may arise in servers that need to make use of external resources for certain applications but not for others. For example, if a single server is handling two applications, one of which uses an external database while the other does not, it may become overloaded for the application that uses the external database when the database response latency increases.

The indication of scopes for overload information (using the Overload-Info-Scope AVP; see Section 5.4) allows a node to indicate a subset of requests to which overload information is to be applied. This document defines seven scopes; only "Connection" scope is mandatory to implement. The use of the optional scopes, along with the use of any additional scopes defined in other documents, is negotiated at connection establishment time; see Section 3.1.

2.1. Scope Descriptions

Destination-Realm: This scope, which nodes MUST implement, pertains to all transactions that have a Destination-Realm AVP matching the indicated value.

Application-ID: This scope, which nodes MUST implement, pertains to all transactions that contain an Application-ID field matching the indicated value.

Destination-Host: This scope, which nodes SHOULD implement, pertains to all transactions that have a Destination-Host AVP matching the indicated value.

Host: This scope, which nodes SHOULD implement, pertains to all transactions sent directly to the host matching the indicated value.

Connection: This scope, which nodes MUST implement, pertains to all transactions sent on the same TCP connection or SCTP association. This scope has no details indicating which connection or association it applies to; instead, the recipient of an indication of "Connection" scope is to use the connection or association on which the message was received as the indicated connection or association. In other words, any use of Connection scope applies to "this connection."

Session-Group: This scope, which nodes MAY implement, pertains to all transactions in a session that has been assigned to the indicated group. For more information on assigning sessions to groups, see Section 3.6.

Session: This scope, which nodes MAY implement, pertains to all transactions in the indicated session.

Some applications do not have long-running sessions containing multiple transactions. For such applications, the use of "Session-Group" and "Session" scopes do not make sense. Such applications will instead make use of the most applicable of the remaining scopes (plus any negotiated extension scopes) to achieve overload control.

OPEN ISSUE: Is there value to including a stream-level scope for SCTP? We haven't been able to come up with a use case for doing so yet, but it wouldn't necessarily be unreasonable.

2.2. Combining Scopes

To allow for the expression of more complicated scopes than the primitives defined above, multiple Overload-Info-Scope AVPs may be included in a single Load-Info AVP. Semantically, these scopes are included in the following way:

- Attributes of the different kinds are logically and-ed together (e.g., if both "Destination-Realm" and "Application-ID" are present, the information applies to requests sent that match both the realm and the application).

- Attributes of the same kind are logically or-ed together (e.g., if two "Destination-Realm"s are present, the information applies to requests sent to either realm).

- If a transaction falls within more than one scope, the "most overloaded" scope is used for traffic shaping.

To prevent the complexity of implementing arbitrary scope combination rules, only the following combinations of scopes are allowed (OPEN ISSUE -- we need to figure out what makes most sense for expressing these combinations. Formal grammar? Prose? A table of some kind? For now, they're expressed as a pseudo-ABNF):

- 1*(Destination-Realm) 0*1(Application-ID)
- 1*(Application-ID) 0*1(Destination-Realm)
- 1*(Application-ID) 0*1(Destination-Host)
- 1*(Application-ID) 0*1(Host)
- 1*(Application-ID) 0*1(Connection)
- 1*(Destination-Host)
- 1*1(Host)
- 1*1(Connection)
- 1*(Session-Group) 0*1(Host | Connection)
- 1*(Session) 0*1(Host | Connection)

OPEN ISSUE: Is this the right set of scope combinations? Is there a need for more? Are any of these unnecessary? Ideally, this should be the smallest set of combinations that lets nodes report what they realistically need to report.

Any document that creates additional scopes MUST define how they may be combined with all scopes registered with IANA at the time of their publication.

3. Diameter Node Behavior

The following sections outline the behavior expected of Diameter clients, servers, and agents that implement the overload control mechanism.

OPEN ISSUE: SIP Overload Control includes a sequence parameter to ensure that out-of-order messages do not cause the receiver to act on state that is no longer accurate. Is message reordering a concern in Diameter? That is, do we need to include sequence numbers in the messages to ensure that the receiver does not act on stale state information? Because Diameter uses only reliable, in-order transports, it seems that this isn't likely to be an issue. Is there room for a race when multiple connections are in use?

3.1. Connection Establishment Procedures

Negotiation for support of this mechanism is performed during Diameter capabilities exchange. Optional protocol features and extensions to this mechanism are also negotiated at this time. No

provision is provided for renegotiation of mechanism use or extensions during the course of a connection. If peers wish to make changes to the mechanism, they must create a new connection to do so.

The connection initiator includes a Load-Info AVP in the CER (Capabilities-Exchange-Request) message that it sends after establishing the connection. This Load-Info AVP MUST contain a Supported-Scopes AVP and an Overload-Algorithm AVP. The Supported-Scopes AVP includes a comprehensive list of scopes supported that the connection initiator can receive and understand. See Section 5.2 for information on the format of the Supported-Scopes AVP.

The Load-Info AVP in a CER message also MAY contain one or more Overload-Algorithm AVPs. If present, these AVPs indicate every Overload-Algorithm the connection initiator is willing to support for the connection that is being established. If the connection initiator supports only the "Loss" algorithm, it MAY indicate this fact by omitting the Overload-Algorithm altogether.

The Load-Info AVP in a CER message MAY also contain additional AVPs, as defined in other documents, for the purpose of negotiation extensions to the Overload mechanism.

The Diameter node that receives a CER message first examines it for the presence of a Load-Info AVP. If no such AVP is present, the node concludes that the overload control mechanism is not supported for this connection, and no further overload-related negotiation is performed. If the received CER contains a Load-Info AVP, the recipient of that message stores that information locally in the context of the connection being established. It then examines the Overload-Algorithm AVPs, if present, and selects a single algorithm from that list. If no Overload-Algorithm is indicated, then the base "Loss" algorithm is used for the connection. In either case, the recipient of the CER stores this algorithm in the context of the connection.

When a node conformant to this specification sends a Capabilities-Exchange-Answer (CEA) message in answer to a CER that contained a Load-Info AVP, the CEA MUST contain a Load-Info AVP. This Load-Info AVP MUST contain a Supported-Scopes AVP that includes a comprehensive list of scopes supported that the connection initiator can receive and understand. The CEA also contains zero or one Overload-Algorithm AVPs. If present, this Overload-Algorithm MUST match one of the Overload-Algorithm AVPs sent in the CER, and it indicates the overload control algorithm that will be used for the connection. If the CEA contains no Overload-Algorithm, the connection will use the "Loss" algorithm.

When a node receives a CEA message, it examines it for the presence of a Load-Info AVP. If no such AVP is present, the node concludes that the overload mechanism is not supported for this connection. If the received CEA contains a Load-Info AVP, then the recipient extracts the Supported-Scopes information, and stores them locally in the context of the connection being established. It then checks for the presence of an Overload-Algorithm AVP. If present, this AVP indicates the overload control algorithm that will be used for the connection. If absent, then the connection will use the "Loss" algorithm.

If a node receives a CEA message that indicates support for a scope that it did not indicate in its CER or which selects an overload control algorithm that it did not advertise in its CER, then it MUST terminate the connection by sending a DPR with a Disconnect-Cause of NEGOTIATION_FAILURE, (128 [actual value TBD]) indicating that the CEA sender has failed to properly follow the negotiation process described above.

Note that the Supported-Scopes announcement during capabilities exchange is a set of mutual advertisements of which scopes the two nodes are willing to receive information about. It is not a negotiation. It is perfectly acceptable for a node to send information for scopes it did not include in the Supported-Scopes AVP it sent, as long as the recipient indicated support for receiving such a scope. For example, a Diameter agent, during connection establishment with a client, may indicate support for receiving only "Connection" and "Host" scope; however, if the client indicated support for "Application" scope, then the agent is free to send Load-Info AVPs that make use of "Application" scope to the client.

3.2. Diameter Client and Diameter Server Behavior

The following sections describe the behavior that Diameter clients and Diameter servers implement for the overload control mechanism. Behavior at Diameter Agents is described in Section 3.3.

To implement overload control, Diameter nodes need to keep track of three important metrics for each of the scopes for which information has been received: the overload metric for the scope, the period of validity for that overload metric, and the load within that scope. Conceptually, these are data records indexed by the scope to which they apply. In the following sections, we refer to these data records with the term "scope entry." Further, when it is necessary to distinguish between those scope entries referring to the load information received from other nodes and those referring to the load information sent to other nodes, we use the term "remote scope entry" to refer to the information received from other nodes, and "local

scope entry" to refer to that information that is being maintained to send to other nodes.

In order to allow recipients of overload information to perform certain performance optimizations, we also define a new command flag, called 'O'verload. This bit, when set, indicates that the message contains at least one Load-Info AVP with a non-zero Overload-Metric -- in other words, the sending node is overloaded for at least one context. See Section 7.4 for the definition of the 'O'verload bit.

OPEN ISSUE: Is there anything we can do to make this 'O'verload bit even more useful? Perhaps setting it only when the overload value has changed, or changed by a certain amount?

3.2.1. Sending a Request to a Compliant Peer

This section applies only to those requests sent to peers who negotiated use of the overload control mechanism during capabilities exchange. Requests sent over other connections are handled the same as they would in the absence of the overload control mechanism.

Before sending a request, a Diameter node must first determine which scope applies. It does this as follows: first, a next hop host and connection are determined, according to normal Diameter procedures (potentially modified as described in Section 3.5.2). The sending node then searches through its list of remote scope entries (ignoring any whose Period-of-Validity has expired) to determine which ones match the combination of the fields in the current request, the next-hop host, and the selected connection. If none of the matching scope entries are in overload, then the message is handled normally, and no additional processing is required.

As an optimization, a sending node MAY choose to track whether any of its peers are in overload, and to skip the preceding step if it knows that no scopes are in overload.

If one or more matching scope entries are in overload, then the sending node determines which scope is most overloaded. The sending node then sends, drops, or otherwise modifies handling of the request according to the negotiated overload control algorithm, using the Overload-Metric from the selected scope entry as input to the algorithm.

When determining which requests are impacted by the overload control algorithm, request senders MAY take into account the type of message being sent and its contents. For example, messages within an existing session may be prioritized over those that create a new session. The exact rules for such prioritization will likely vary

from application to application. The authors expect that specifications that define or specify the use of specific Diameter Applications may choose to formally define a set of rules for such prioritization on a per-Application basis.

The foregoing notwithstanding, senders MUST NOT use the content or type of request to exempt that request from overload handling. For example, if a peer requests a 50% decrease in sent traffic using the "Loss" algorithm (see Section 4), but the traffic that the sending node wishes to send consists 65% of traffic that the sender considers critical, then the sender is nonetheless obliged to drop some portion of that critical traffic (e.g., it may elect to drop all non-critical traffic and 23% of the critical traffic, resulting in an overall 50% reduction).

The sending node then inserts one or more Load-Info AVPs (see Section 5.1) into the request. If the sender inserts more than one Load-Info AVP, then each Load-Info AVP MUST contain a unique scope, as specified by the Overload-Scope AVP(s) inside the Load-Info AVP.

Each Load-Info AVP in the request MUST contain an Overload-Metric (see Section 5.5), indicating whether (and to what degree) the sender is overloaded for the indicated scope. If this metric is not zero, then the Load-Info AVP MUST also contain a Period-Of-Validity AVP (see Section 5.6), indicating the maximum period the recipient should consider the Overload-Metric to be valid. Any message containing a non-zero Overload-Metric also MUST set the 'O'verload bit in the Command Flags field to indicate to the recipient that the message contains an overload indication. See Section 7.4 for the definition of the 'O'verload bit.

Each Load-Info AVP MUST also contain a Load AVP, indicating the server's load level within the context of the indicated scope. See Section 3.5.1 for details on generating this load metric. Note that a server's load may frequently be identical for all the scopes for which it sends information.

3.2.2. Receiving a Request

3.2.2.1. Receiving a Request from a Compliant Peer

A node that receives a request from a peer that has negotiated support for the overload control mechanism will extract the Load-Info AVPs from the request and use each of them to update its remote scope entries. First, the node attempts to locate an existing scope entry that corresponds to the Overload-Scope indicated in the Load-Info AVP. If one does not exist, it is created. The scope entry is then populated with the overload metric, period of validity, and load

information. The message is then processed as normal.

In some circumstances, request recipients can become sufficiently overloaded that even those messages received from complaint clients can overwhelm its processing capabilities. Under such circumstances, nodes MAY begin treating a subset of such requests as if they were received from noncompliant peers (as explained in the following section).

3.2.2.2. Receiving a Request from a Noncompliant Peer

An important aspect of the overload control mechanism is that Diameter nodes that do not implement the mechanism cannot have an advantage over those that do. In other words, it is necessary to prevent the situation that a network in overload will cease servicing those transactions from overload-compliant nodes in favor of those sent by those nodes that do not implement the overload control mechanism. To achieve this goal, message recipients need to track the overload control metric on behalf of those sending nodes that do not implement overload, and to reject messages from those nodes that would have been dropped if the sender had implemented the overload mechanism.

A node that receives a request from a peer that has not negotiated support for the overload control mechanism searches through its list of local scope entries to determine which ones match the combination of the fields in the received request. (These are the entries that indicate the Overload-Metric that the node would have sent to the peer if the peer had supported the overload mechanism). If none of the matching scope entries are in overload, then the message is sent normally, and no additional processing is required.

If one or more matching local scope entries are in overload, then the node determines which scope is most overloaded. The node then executes the "Loss" overload control algorithm (see Section 4) using the overload metric in that most overloaded scope. If the result of running that algorithm determines that a sender who had implemented the overload control mechanism would have dropped the message, then the recipient MUST reply to the request with a DIAMETER_PEER_IN_OVERLOAD response (see Section 7.3).

3.2.3. Sending an Answer to a Compliant Peer

This section applies only to those answers sent to peers who negotiated use of the overload control mechanism during capabilities exchange.

When sending an answer, a Diameter node inserts one or more Load-Info

AVPs (see Section 5.1) into the answer. If the sender inserts more than one Load-Info AVP, then each Load-Info AVP MUST contain a unique scope, as specified by the Overload-Scope AVP(s) inside the Load-Info AVP.

Each Load-Info AVP in the answer MUST contain an Overload-Metric (see Section 5.5), indicating whether (and to what degree) the server is overloaded for the indicated scope. If this metric is not zero, then the Load-Info AVP MUST also contain a Period-Of-Validity AVP (see Section 5.6), indicating the maximum period the recipient should consider the Overload-Metric to be valid. Any message containing a non-zero Overload-Metric also MUST set the 'O'verload bit in the Command Flags field to indicate to the recipient that the message contains an overload indication. See Section 7.4 for the definition of the 'O'verload bit.

It is important to note that using this mechanism creates a closed feedback loop, with some amount of lag introduced by overload processing and the network. As such, implementers must be aware of the potential for such a system to produce oscillations in the overload level. Without proper control, it is also possible for these oscillations to diverge, resulting in undesirable behavior. There are several ways to address this issue, and it is left to implementors to determine the best way for their particular situation. However, at a minimum senders of overload control information SHOULD apply hysteresis to the Overload-Metric, and signal easing of overload more slowly than signaling increases.

Each Load-Info AVP MUST also contain a Load AVP, indicating the server's load level within the context of the indicated scope. See Section 3.5.1 for details on generating this load metric. Note that a server's load may frequently be identical for all the scopes for which it sends information.

3.2.4. Receiving an Answer from a Compliant Peer

A node that receives an answer from a peer that has negotiated support for the overload control mechanism will extract the Load-Info AVPs from the answer and use each of them to update its remote scope entries. First, the node attempts to locate an existing scope entry that corresponds to the Overload-Scope indicated in the Load-Info AVP. If one does not exist, it is created. The scope entry is then populated with the overload metric, period of validity, and load information. The message is then processed as normal.

3.3. Diameter Agent Behavior

This section discusses the behavior of a Diameter Agent acting as a Proxy or Relay. Diameter Agents that provide redirect or translation services behave the same as Diameter Servers for the purpose of overload control, and follow the procedures defined in Section 3.2.

Whenever sending a request or an answer, Agents MUST include a Load-Info AVP reflecting the Agent's overload and load information. In formulating this information, the Agent may choose to use only that information relating to its own local resources. However, better network behavior can be achieved if agents incorporate information received from their peers when generating overload information. The exact means for incorporating such information is left to local policy at the agent.

For example: consider an agent that distributes sessions and transactions among three Diameter servers, each hosting a different Diameter application. While it would be compliant for the Agent to only report its own overload state (i.e., at "Host" scope), overall network behavior would be improved if it chose to also report overload state for up to three additional scopes (i.e. at "Application-ID" scope), incorporating the Overload information received from each server in these scopes.

3.3.1. Proxying a Request

Upon receiving a request, a Diameter Proxy or Relay performs the steps detailed in Section 3.2.2.

The agent then MUST remove all Load-Info AVPs from the request: Load-Info is never passed through a Proxy or Relay transparently.

When the Diameter Agent proxies or relays a request, it follows the process outlined in Section 3.2.1.

3.3.2. Proxying an Answer

Upon receiving an answer, a Diameter Agent follows the process described in Section 3.2.4 to update its remote scope entries.

The Agent then MUST remove all Load-Info AVPs from the answer: Load-Info is never passed through a Proxy or Relay transparently.

When the Diameter Agent proxies or relays a response, it follows the process outlined in Section 3.2.3.

3.4. Proactive Load and Overload Communication

Because not all Diameter links will have constant traffic, it may be occasionally necessary to send overload and/or load information over links that would otherwise be quiescent. To proactively send such information to peers, the Diameter node with information to convey may choose to send a Diameter Watchdog Request (DWR) message to its peers. The procedure described in Section 3.2.1 applies to these requests, which provides the means to send load and overload information.

In order to prevent unnecessarily diminished throughput between peers, a Diameter node SHOULD proactively send a DWR to all its peers whenever it leaves an overload state. Similarly, in order to provide peers the proper data for load distribution, nodes SHOULD send DWR messages to a peer if the load information most recently sent to that peer has changed by more than 20% and is more than 5 seconds old.

3.5. Load Processing

While the remainder of the mechanism described in this document is aimed at handling overload situations once they occur, it is far better for a system if overload can be avoided altogether. In order to facilitate overload avoidance, the overload mechanism includes the ability to convey node load information.

Semantically, the Load information sent by a Diameter node indicates the current utilization of its most constrained resource. It is a linear scale from 0 (least loaded) to 65535 (most loaded).

It is critical to distinguish between the value conveyed in the Load AVP and the value conveyed in the Overload-Metric AVP. The Load AVP is computed and used independent of the Overload-Algorithm selected for a connection, while the Overload-Metric is meaningful only in the context of the selected algorithm. Most importantly, the Load information never has any impact on the behavior specified in the overload algorithm. If a node reports a Load of 65535, but the Overload-Metric does not indicate any need to apply the selected overload control algorithm, then the sender MUST NOT apply the selected overload control algorithm. Conversely, if a node is reporting an Overload-Metric that requires the recipient to take action to reduce traffic, those actions MUST be taken, even if the node is simultaneously reporting a Load value of 0.

3.5.1. Sending Load Information

Diameter nodes implementing the overload mechanism described in this document MUST include a Load AVP (inside a Load-Info AVP) in every

Diameter message (request and answer) they send over a connection that has been negotiated to use the overload control mechanism. Note that this requirement does not necessitate calculation of the Load metric each time a message is sent; the Load value may be calculated periodically (e.g., every 100 ms), and used for every message sent until it is recalculated.

The algorithm for generation of the load metric is a matter of local policy at the Diameter node, and may vary widely based on the internal software architecture of that node.

For advanced calculations of Load, anticipated inputs to the computation include CPU utilization, network utilization, processor interrupts, I/O throughput, and internal message queue depths.

To free implementors from the potential complexity of determining an optimal calculation for load, we define a very simple, baseline load calculation that MAY be used for the purpose of populating the Load AVP. Implementations using this simplified calculation will use a configured, hard-coded, or Service Level Agreement (SLA)-defined maximum number of transactions per second (TPS) which a node is known to be able to support without issue. These implementations simply report their load as a linear representation of how much of this known capacity is currently in use:

```
Load = MIN(Current_TPS * 65535 / Maximum_TPS, 65535)
```

To prevent rapid fluctuations in the load metric, nodes SHOULD report a rolling average of the calculated load rather than the actual instantaneous load at any given moment.

Load information is scoped to the level indicated by the Overload-Info-Scope AVP present in the Load-Info AVP in which the Load AVP appears.

3.5.2. Receiving Load Information

While sending load information is mandatory, the actual processing of load information at a recipient is completely optional. Ideally, recipients will use the load information as input to a decision regarding which of multiple equivalent servers to use when initiating a new connection. Recipients may choose to update load information on receipt of every message; alternately, they may periodically "sample" messages from a host to determine the load it is currently reporting.

3.5.2.1. Example Load Handling

This section describes a non-normative example of how recipients can use Load information received from other Diameter nodes. At a high level, the concept is that received load metrics are used to scale the distribution algorithm that the node uses for selection of a server from a group of equivalent servers.

Consider a client that uses DNS to resolve a host name into IP addresses. In this example, the client is attempting to reach the server for the realm example.com. It performs a NAPTR query for the "AAA+D2T" record for that domain, and receives a result pointing to the SRV record "_diameter._tcp.example.com". Querying for this SRV record, in turn, results in three entries, with the same priorities:

SRV Weight	Server Name
20	server-a.example.com
20	server-b.example.com
60	server-c.example.com

The client then examines the currently reported loads for each of the three servers. In this example, we are asserting that the reported load metrics are as follows:

Load	Server Name
13107 (20%)	server-a.example.com
26214 (60%)	server-b.example.com
52428 (80%)	server-c.example.com

Based on this load information, the client scales the SRV weights proportional to each server's reported load; the general formula is:

```
new_weight = original_weight * (65535 - load) / 65535
```

The node then calculates a new set of weights for the destination hosts:

- server-a: new_weight = 20 * (65535 - 13107) / 65535 = 16
- server-b: new_weight = 20 * (65535 - 26214) / 65535 = 12
- server-c: new_weight = 60 * (65535 - 52428) / 65535 = 12

These three new weights (16, 12, and 12) are then used as input to

the random selection process traditionally used when selecting among several SRV records.

Note that this example is provided in the context of DNS SRV processing; however, it works equally well in the case that server processing weights are provisioned or made available through an alternate resolution process.

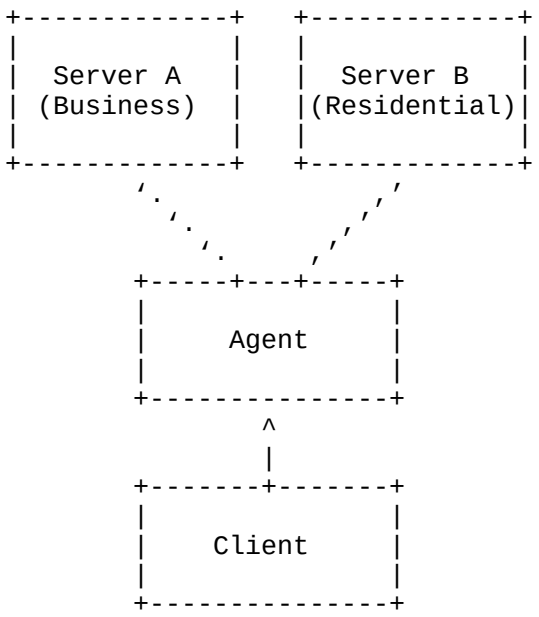
3.6. Session Establishment for Session Groups

The procedure in this section applies to any Diameter operation that may result in the creation of a new Diameter session. Note that these operations are performed in addition to any normal message processing, and in addition to the operations described in the following sections.

3.6.1. Session Group Concepts

At the time a session is established, the server and/or the client may choose to assign the newly created session to a Session Group that they can use to refer to the session (and other sessions in the same group) in later overload-related messages. This grouping is intended to be used by servers that have visibility into resources that may be independently overloaded, but which do not correspond to an existing Diameter construct (such as Application, Realm, or Destination Server).

One example of a server having visibility into resources that don’t have a corresponding Diameter construct is a Diameter Agent servicing a mixed community of users -- say, one authenticated by a "Business" server, and another authenticated by a "Residential" server. The client in this network does not know which group any given session belongs in; the routing of sessions is based on information available only to the agent.



In this case, the Agent may wish to assign sessions to two client-visible Session Groups when the session is established. By doing so, the Agent gains the ability to report Load and Overload metrics to the Client independently for the two classes of users. This can be extremely helpful, for example, in allowing the Agent to ask the Client to throttle traffic for the Residential server when it becomes overload, without impacting sessions pertaining to the Business server.

Similar situations can arise even without the presence of Diameter Agents in the network: a server may have a class of sessions that require access to an off-board database (which can, itself, become overloaded), while also servicing a class of sessions that is handled entirely by a local authentication table. The server can use Session Groups to assign these two classes of sessions to different groups, and report overload on the class using the (overloaded) off-board database without impacting the other sessions.

In some applications, it is possible to have the session established by one peer (e.g., in the upstream direction), while some subsequent in-session transactions are initiated by the other peer (e.g., in the downstream direction). Because of this possibility, the overload mechanism allows both peers to establish a Session Group at the time the session is set up. The session identifiers are scoped to the node that sends them. In other words, if a server assigns a session to a group called "Residential", this group is not related to a

client group (if any) by the same name. For clarity, this document will refer to the session group assigned by the server performing the processing as a "local session group," and the session group assigned by the remote node as a "remote session group."

Nodes that send a session-creating request follow normal Diameter procedures, along with the additional behavior described in Section 3.2.1 and Section 3.3.1, as appropriate. Such nodes may also assign the session to a Session Group, as long as the peer to which they are communicating indicated support for the "Session-Group" scope during capabilities exchange. Whether to do so and what group to assign a session to is done according to local policy. To perform such assignment, the node will include a Session-Group AVP (see Section 5.7 in the Load-Info AVP for the session creating request. These nodes also store the assigned name as the session's local session group.

3.6.2. Session Group Procedures

The procedures in this section only apply on connections for which support for the "Session-Group" scope has been negotiated during capabilities exchange. See Section 3.1.

When a node receives a session creating request, it MUST check that request for the presence for a Session-Group AVP in its Load-Info AVP. If one is present, it stores that session group name as the remote session group name for that server. This allows clients to assign the session to a group, allowing it to indicate overload for server-initiated transactions in the resulting session.

When a node replies to a session creating request, it can choose to assign the newly-established session to a session group. Whether it chooses to do so is independent of whether the remote node assigned the session to a session group. To perform such an assignment, the node includes a Session-Group AVP in the Load-Info AVP sent in answer to the session-creating request. These nodes also store the assigned name as the session's local session group.

Finally, when a node that has sent a session-creating request receives a corresponding answer message, it MUST check that answer for the presence of a Session-Group AVP in its Load-Info AVP. If one is present, it stores that session group name as the remote session group name for that server.

4. Loss-Based Overload Control Algorithm

This section describes a baseline, mandatory-to-implement overload

control algorithm, identified by the indicator "Loss". This algorithm allows a Diameter peer to ask its peers to reduce the number of requests they would ordinarily send by a specified percentage. For example, if a peer requests of another peer that it reduce the traffic it is sending by 10%, then that peer will redirect, reject, or treat as failed, 10% of the traffic that would have otherwise been sent to this Diameter node.

4.1. Overload-Metric values for the 'Loss' Algorithm

A Diameter node entering the overload state for any of the scopes that it uses with its peers will calculate a value for its Overload Metric, in the range of 0 to 100 (inclusive). This value indicates the percentage traffic reduction the Diameter node wishes its peers to implement. The computation of the exact value for this parameter is left as an implementation choice at the sending node. It is acceptable for implementations to request different levels of traffic reduction to different peers according to local policy at the Diameter node. These Overload Metrics are then communicated to peers using the Overload-Metric AVP in requests and answers sent by this node.

Recipients of Overload-Metric AVPs on connections for which the "Loss" algorithm has been specified MUST reduce the number of requests sent in the corresponding scope by that percentage, either by redirecting them to an alternate destination, or by failing the request. For a Diameter Agent, these failures are indicated to the peer who originated the request by sending a DIAMETER_PEER_IN_OVERLOAD response (see Section 7.3). For diameter clients, these failures cause the client to behave as if they received a transient error in response to the request.

It is acceptable, when implementing the "Loss" algorithm, for the reduction in transactions to make use of a statistical loss function (e.g., random assignment of transactions into "success" and "failure" categories based on the indicated percentage). In such a case, the actual traffic reduction might vary slightly from the percentage indicated, albeit in an insignificant amount.

The selection of which messages to withhold from sending does not need to be arbitrary. For example, implementations are allowed to distinguish between higher-priority and lower-priority messages, and drop the lower-priority messages in favor of dropping the higher priority messages, as long as the total reduction in traffic conforms to the Overload-Metric in effect at the time. The selection of which messages to prioritize over others will likely vary from application to application (and may even be subject to standardization as part of the application definition). One example of such a prioritization

scheme would be to treat those messages that result in the creation of a new session as lower priority then those messages sent in the context of an established session.

4.2. Example Implementation

The exact means a client uses to implement the requirement that it reduce traffic by a requested percentage is left to the discretion of the implementor. However, to aid in understanding the nature of such an implementation, we present an example of a valid implementation in pseudo-code.

In this example, we consider that the sending node maintains two classes of request. The first category are considered of lower priority than the second category. If a reduction in traffic is required, then these lower priority requests will be dropped before any of the higher priority requests are dropped.

The sending Diameter node determines the mix of requests falling into the first category, and those falling into the second category. For example, 40% of the requests may be in the lower-priority category, while 60% are in the higher-priority category.

When a node receives an overload indication from one of its peers, it converts the Overload-Metric value to a value that applies to the first category of requests. For example, if the Overload-Metric for the applicable context is "10", and 40% of the requests are in the lower-priority category, then:

```
10 / 40 * 100 = 25
```

Or 25% of the requests in the first category can be dropped, with an overall reduction in sent traffic of 10%. The sender then drops 25% of all category 1 requests. This can be done stochastically, by selecting a random number for each sent packet between 1 to 100 (inclusive), and dropping any packet for which the resulting percentage is equal to or less than 25. In this set of circumstances, messages in the second category do not require any reduction to meet the requirement of 25% traffic reduction.

A reference algorithm is shown below, using pseudo-code.

```
cat1 := 80.0         // Category 1 --- subject to reduction
cat2 := 100.0 - cat1 // Category 2 --- Under normal operations
// only subject to reduction after category 1 is exhausted.
// Note that the above ratio is simply a reasonable default.
// The actual values will change through periodic sampling
// as the traffic mix changes over time.
```

```
   while (true) {
     // We're modeling message processing as a single work queue
     // that contains both incoming and outgoing messages.
     msg := get_next_message_from_work_queue()

     update_mix(cat1, cat2)  // See Note below

     switch (msg.type) {

       case outbound request:
         destination := get_next_hop(msg)
         oc_context := get_oc_scope(destination,msg)

         if (we are in overload) {
           add_overload_avps(msg)
         }

         if (oc_context == null)  {
             send_to_network(msg) // Process it normally by sending the
             // request to the next hop since this particular
             // destination is not subject to overload
         }
         else  {
            // Determine if server wants to enter in overload or is in
            // overload
            in_oc := extract_in_oc(oc_context)

            oc_value := extract_oc(oc_context)
            oc_validity := extract_oc_validity(oc_context)

            if (in_oc == false or oc_validity is not in effect)  {
               send_to_network(msg) // Process it normally by sending
               // the request to the next hop since this particular
               // destination is not subject to overload.  Optionally,
               // clear the oc context for this server (not shown).
            }
            else  {  // Begin perform overload control
               r := random()
               drop_msg := false

               if (cat1 >= cat2) {
                   category := assign_msg_to_category(msg)
                   pct_to_reduce_cat2 := 0
                   pct_to_reduce_cat1 := oc_value / cat1 * 100
                   if (pct_to_reduce_cat1 > 100)  {
                      // Get remaining messages from category 2
                      pct_to_reduce_cat2 := 100 - pct_to_reduce_cat1
                      pct_to_reduce_cat1 := 100
```

```
                }

                if (category == cat1)  {
                   if (r <= pct_to_reduce_cat1)  {
                      drop_msg := true
                   }
                }
                else {  // Message from category 2
                   if (r <= pct_to_reduce_cat2)  {
                      drop_msg := true
                   }
                }
             }
             else  { // More category 2 messages than category 1;
                     // indicative of an emergency situation.  Since
                     // there are more category 2 messages, don't
                     // bother distinguishing between category 1 or
                     // 2 --- treat them equal (for simplicity).
                if (r <= oc_value)
                   drop_msg := true
             }

             if (drop_msg == false) {
                 send_to_network(msg) // Process it normally by
                // sending the request to the next hop
             }
             else  {
                // Do not send request downstream, handle locally by
                // generating response (if a proxy) or treating as
                // an error (if a user agent).
             }
          }  // End perform overload control
        }

      end case // outbound request

      case outbound answer:
        if (we are in overload) {
          add_overload_avps(msg)
        }
        send_to_network(msg)

      end case // outbound answer

      case inbound answer:
         create_or_update_oc_scope()  // For the specific server
         // that sent the answer, create or update the oc scope;
         // i.e., extract the values of the overload AVPs
```

```
         // and store them in the proper scopes for later use.
         process_msg(msg)

      end case // inbound answer
      case inbound request:
         create_or_update_oc_scope()

        if (we are not in overload)  {
           process_msg(msg)
        }
        else {  // We are in overload
           if ( connection supports overload)
              process_msg(msg)
           }
           else {  // Sender does not support oc
              if (local_policy(msg) says process message)  {
                 process_msg(msg)
              }
              else  {
                 send_answer(msg, DIAMETER_PEER_IN_OVERLOAD)
              }
           }
        }
      end case // inbound request
    }
  }
```

A simple way to sample the traffic mix for category 1 and category 2 is to associate a counter with each category of message. Periodically (every 5-10s), get the value of the counters and calculate the ratio of category 1 messages to category 2 messages since the last calculation.

Example: In the last 5 seconds, a total of 500 requests were scheduled to be sent. Assume that 450 out of 500 were messages subject to reduction and 50 out of 500 were classified as requests not subject to reduction. Based on this ratio, cat1 := 90 and cat2 := 10, or a 90/10 mix will be used in overload calculations.

Of course, this scheme can be generalized to include an arbitrary number of priorities, depending on how many different classes of messages make sense for the given application.

5. Diameter AVPs for Overload

NOTE: THE AVP NUMBERS IN THIS SECTION ARE USED FOR EXAMPLE PURPOSES ONLY. THE FINAL AVP CODES TO BE USED WILL BE ASSIGNED BY IANA DURING THE PUBLICATION PROCESS, WHEN AND IF THIS DOCUMENT IS PUBLISHED AS AN RFC.

Attribute Name	AVP Code	Sec. Def.	Data Type	MUST	MUST NOT
Load-Info	1600	5.1	Grouped		M,V
Supported-Scopes	1601	5.2	Unsigned64		M,V
Overload-Algorithm	1602	5.3	Enumerated		M,V
Overload-Info-Scope	1603	5.4	OctetString		M,V
Overload-Metric	1604	5.5	Unsigned32		M,V
Period-Of-Validity	1605	5.6	Unsigned32		M,V
Session-Group	1606	5.7	UTF8String		M,V
Load	1607	5.8	Unsigned32		M,V

5.1. Load-Info AVP

The Load-Info AVP (AVP code 1600) is of type Grouped, and is used as a top-level container to group together all information pertaining to load and overload information. Every Load-Info AVP MUST contain one Overload-Information-Scope AVP, and one Overload-Metric AVP.

The Grouped Data field of the Load-Info AVP has the following CCF grammar:

```
< Load-Info > ::= < AVP Header: 1600 >
                  < Overload-Metric >
                * { Overload-Info-Scope }
                  [ Supported-Scopes ]
                * [ Overload-Algorithm ]
                  [ Period-Of-Validity ]
                  [ Session-Group ]
                  [ Load ]
                * [ AVP ]
```

5.2. Supported-Scopes AVP

The Supported-Scopes AVP (AVP code 1601) is of type Uint64, and is used during capabilities exchange to indicate the scopes that a given node can receive on the connection. Nodes that support the mechanism defined in this document MUST include a Supported-Scopes AVP in all CER messages. It also MUST appear in any CEA messages sent in answer

to a CER message containing a Load-Info AVP. The Supported-Scopes AVP MUST NOT appear in any other message types. See Section 5.4 for an initial list of scopes.

The Supported-Scopes AVP contains a bitmap that indicates the scopes supported by the sender. Within the bitmap, the least significant bit indicates support for scope 1 (Destination-Realm), while the next least significant bit indicates support for scope 2 (Application-ID), and so on. In general, if we consider the bits to be numbered from 0 (LSB) to 63 (MSB), then any bit n corresponds to the scope type numbered n+1. This scheme allows for up to 64 total scopes to be supported. More formally, the bitmask used to indicate support for any specific context is calculated as follows (where the symbol "<<" indicates a bit shift left):

```
bitmask = 1 << (n - 1)
```

For additional clarity, the bitmasks for the scopes defined in this document are as follows:

Scope	Bitmask	Scope
1	0x0000000000000001	Destination-Realm
2	0x0000000000000002	Application-ID
3	0x0000000000000004	Destination-Host
4	0x0000000000000008	Host
5	0x0000000000000010	Connection
6	0x0000000000000020	Session-Group
7	0x0000000000000040	Session

The advertisement process that makes use of the Supported-Scopes AVP is described in Section 3.1.

5.3. Overload-Algorithm AVP

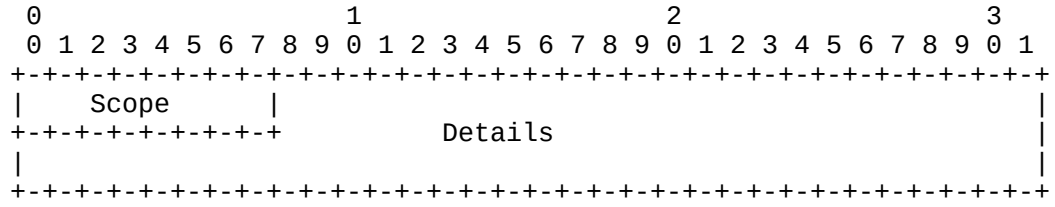
The Overload-Algorithm AVP (AVP code 1602) is of type Enumerated, and is used to negotiate the algorithm that will be used for load abatement. The Overload-Algorithm AVP MAY appear in CER and CEA messages, and MUST NOT appear in any other message types. If absent, an Overload Algorithm of type 1 (Loss) is indicated. Additional values can be registered by other documents; see Appendix C.1. Initial values for the enumeration are as follows:

AVP Values	Attribute Name	Reference
0	Reserved	-
1	Loss	[RFC xxxx]

5.4. Overload-Info-Scope AVP

The Overload-Info-Scope AVP (AVP code 1603) is of type OctetString, and is used to indicate to which scope the Overload-Metric applies.

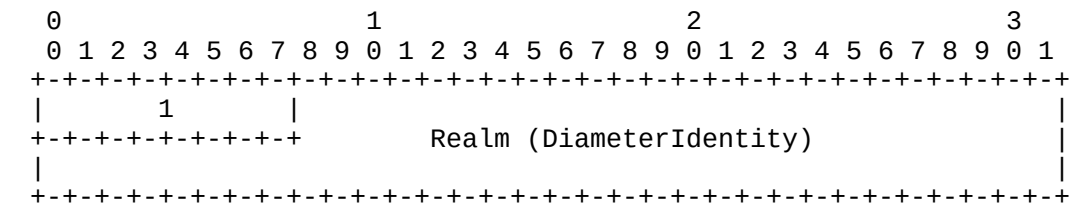
See Section 2 for a definition of the different scope types and a formal description of how they are applied. Other documents may define additional scopes; see Appendix C.2 for details.



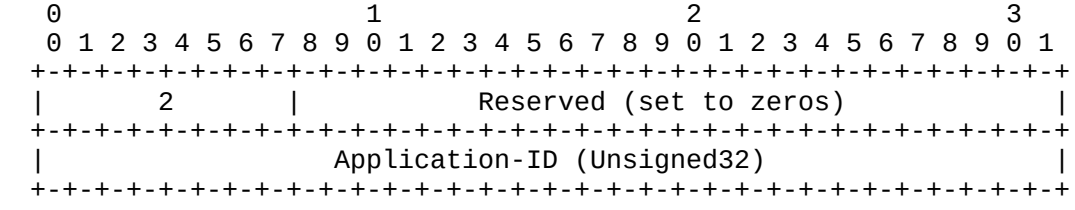
Scope	Attribute Name	Reference
0	Reserved	[RFC xxxx]
1	Destination-Realm	[RFC xxxx]
2	Application-ID	[RFC xxxx]
3	Destination-Host	[RFC xxxx]
4	Host	[RFC xxxx]
5	Connection	[RFC xxxx]
6	Session-Group	[RFC xxxx]
7	Session	[RFC xxxx]

Each Overload-Info-Scope has a different encoding, according to the identifier used to designate the corresponding scope. The formats for the seven scopes defined in this document are given in the following section.

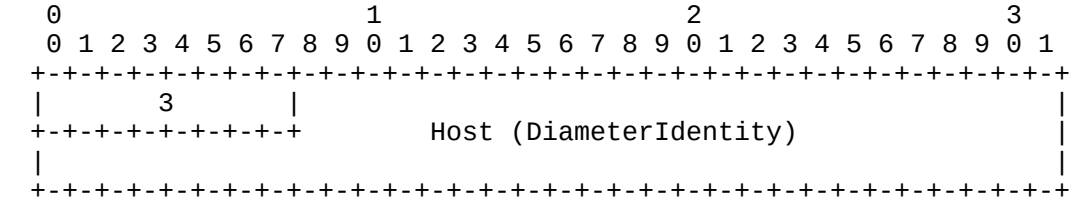
5.4.1. Realm Scope



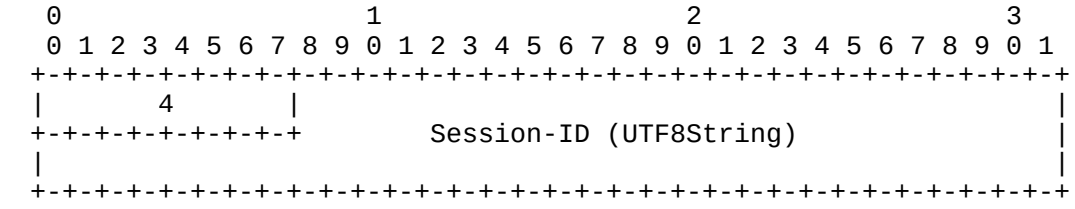
5.4.2. Application-ID Scope



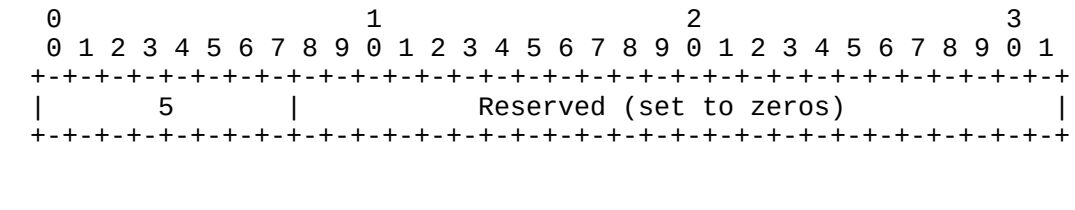
5.4.3. Host Scope



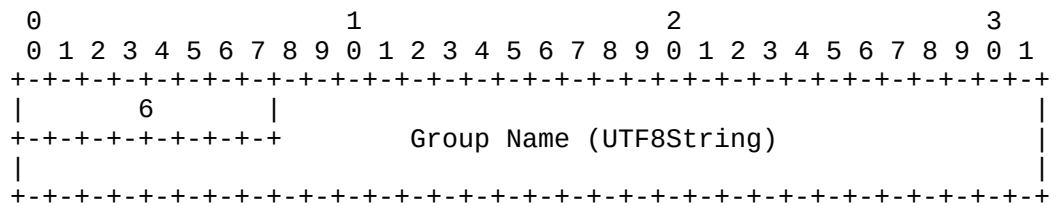
5.4.4. Session Scope



5.4.5. Connection Scope



5.4.6. Session Group Scope



5.5. Overload-Metric AVP

The Overload-Metric AVP (AVP code 1604) is of type Unsigned32, and is used as input to the load mitigation algorithm. Its definition and interpretation is left up to each individual algorithm, with the exception that an Overload-Metric of "0" always indicates that the node is not in overload (that is, no load abatement procedures are in effect) for the indicated scope.

5.6. Period-Of-Validity AVP

The Period-Of-Validity AVP (AVP code 1605) is of type Unsigned32, and is used to indicate the length of time, in seconds, the Overload-Metric is to be considered valid (unless overridden by a subsequent Overload-Metric in the same scope). It MUST NOT be present if the Overload-Metric is ’0’, and MUST be present otherwise.

5.7. Session-Group AVP

The Session-Group AVP (AVP code 1606) is of type UTF8String, and is used to assign a new session to the session group that it names. The Session-Group AVP MAY appear once in the answer to a session-creating request, and MUST NOT appear in any other message types.

5.8. Load AVP

The Load AVP (AVP code 1607) is of type Unsigned32, and is used to indicate the load level of the scope in which it appears. See Section 3.5 for additional information.

6. Security Considerations

A key concern for recipients of overload metrics and load information is whether the peer from which the information has been received is authorized to speak for the indicated scope. For scopes such as "Host" and "Connection", such authorization is obvious. For other scopes, such as "Application-ID" and "Realm", the potential for a

peer to maliciously or accidentally reduce traffic to a third party is evident. Implementations may choose to ignore indications from hosts which do not clearly have authority over the indicated scope; alternately, they may wish to further restrict the scope to apply only to the host from which the information has been received.

On the other hand, multiple nodes that are under the same administrative control (or a tightly controlled confederation of control) may be implicitly trusted to speak for all scopes within that domain of control. Implementations are encouraged to allow configuration of inherently trusted servers to which the foregoing restrictions are not applied.

Open Issue: There are almost certainly other security issues to take into consideration here. For example, we might need to include guidance around who gets to see our own load information, and potentially changing the granularity of information presented based on trust relationships.

7. IANA Considerations

This document defines new entries in several existing IANA tables. It also creates two new tables.

7.1. New Diameter AVPs

The following entries are added to the "AVP Codes" table under the "aaa-parameters" registry.

AVP Code	Attribute Name	Reference
1600	Load-Info	RFC xxxx
1601	Supported-Scopes	RFC xxxx
1602	Overload-Algorithm	RFC xxxx
1603	Overload-Info-Scope	RFC xxxx
1604	Overload-Metric	RFC xxxx
1605	Period-Of-Validity	RFC xxxx
1606	Session-Group	RFC xxxx
1607	Load	RFC xxxx

7.2. New Diameter Disconnect-Cause

The following entry is added to the "Disconnect-Cause AVP Values (code 273)" table in the "aaa-parameters" registry:

AVP Values	Attribute Name	Reference
128 [actual value TBD]	NEGOTIATION_FAILURE	RFC xxxx

7.3. New Diameter Response Code

The following entry is added to the "Result-Code AVP Values (code 268) - Transient Failures" table in the "aaa-parameters" registry:

AVP Values	Attribute Name	Reference
4128 [actual value TBD]	DIAMETER_PEER_IN_OVERLOAD	RFC xxxx

7.4. New Command Flag

The following entry is added to the "Command Flags" table in the "aaa-parameters" registry:

bit	Name	Reference
4	'O'verload	RFC xxxx

7.5. Overload Algorithm Registry

This document defines a new table, to be titled "Overload-Algorithm Values (code 1602)", in the "aaa-parameters" registry. Its initial values are to be taken from the table in Section 5.3.

New entries in this table follow the IANA policy of "Specification Required." (Open Issue: The WG should discuss registration policy to ensure that we think this is the right balance).

7.6. Overload Scope Registry

This document defines a new table, to be titled "Overload-Info-Scope Values (code 1603)", in the "aaa-parameters" registry. Its initial values are to be taken from the table in Section 5.4.

New entries in this table follow the IANA policy of "Specification Required." (Open Issue: The WG should discuss registration policy to ensure that we think this is the right balance).

8. References

8.1. Normative References

[I-D.ietf-dime-overload-reqs]
McMurry, E. and B. Campbell, "Diameter Overload Control Requirements", draft-ietf-dime-overload-reqs-06 (work in progress), April 2013.

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

[RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.

8.2. Informative References

[I-D.ietf-soc-overload-control]
Gurbani, V., Hilt, V., and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control", draft-ietf-soc-overload-control-12 (work in progress), February 2013.

[I-D.ietf-soc-overload-rate-control]
Noel, E. and P. Williams, "Session Initiation Protocol (SIP) Rate Control", draft-ietf-soc-overload-rate-control-04 (work in progress), April 2013.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[RFC6357] Hilt, V., Noel, E., Shen, C., and A. Abdelal, "Design Considerations for Session Initiation Protocol (SIP) Overload Control", RFC 6357, August 2011.

Appendix A. Acknowledgements

This work was inspired by and borrows heavily from the SIP overload control mechanism described in [I-D.ietf-soc-overload-control]. The authors of this document are deeply grateful to the editor and authors of that work, as well as its many contributors.

Thanks to Ben Campbell for significant input to the initial mechanism design. The author also thanks Martin Dolly, Bob Wallace, John Gilmore, Matt McCann, Jonathan Palmer, Kedar Karmarkar, Imtiaz Shaikh, Jouni Korhonen, Uri Baniel, Jianrong Wang, Brian Freeman, and

Eric Noel for early feedback on the mechanism.

Appendix B. Requirements Analysis

This section analyzes the mechanism described in this document against the set of requirements detailed in [I-D.ietf-dime-overload-reqs].

REQ 1: The overload control mechanism MUST provide a communication method for Diameter nodes to exchange load and overload information.

> Compliant. The mechanism uses new AVPs piggybacked on existing Diameter messages to exchange load and overload information.

REQ 2: The mechanism MUST allow Diameter nodes to support overload control regardless of which Diameter applications they support. Diameter clients must be able to use the received load and overload information to support graceful behavior during an overload condition. Graceful behavior under overload conditions is best described by REQ 3.

> Compliant. Piggybacked AVPs conveying overload control information is sent on every Diameter message to compliant peers, without regard to its Application-ID. The use of the Application-ID scope allows information relevant to one application to be piggybacked on messages for other applications.
>
> Information sent to peers includes load and overload information for use by overload control algorithms, intended for graceful overload mitigation. The mechanism is hop-by-hop and has provisions for agents to forward or aggregate load and overload information towards clients and servers so that each element can have appropriate information for graceful overload control.

REQ 3: The overload control mechanism MUST limit the impact of overload on the overall useful throughput of a Diameter server, even when the incoming load on the network is far in excess of its capacity. The overall useful throughput under load is the ultimate measure of the value of an overload control mechanism.

Compliant. The mechanism provides information nodes use to affect the impacts of overload according to agreed upon algorithms. By controlling or reducing traffic sent towards overloaded elements, using overload control information as described in the mechanism, the effects of overload can be limited. Use of scopes provides a means to minimize the impact of overload mitigation, increasing overall useful throughput during overload conditions.

REQ 4: Diameter allows requests to be sent from either side of a connection and either side of a connection may have need to provide its overload status. The mechanism MUST allow each side of a connection to independently inform the other of its overload status.

Compliant. Overload control information can be piggybacked on any Diameter message. This applies for requests and answers sent from either side of a connection.

REQ 5: Diameter allows nodes to determine their peers via dynamic discovery or manual configuration. The mechanism MUST work consistently without regard to how peers are determined.

Compliant. The mechanism makes no assumptions as to how peers are determined. Discovery of supporting peers is accomplished as part of the normal capabilities exchange and does not affect how or where these exchanges occur.

REQ 6: The mechanism designers SHOULD seek to minimize the amount of new configuration required in order to work. For example, it is better to allow peers to advertise or negotiate support for the mechanism, rather than to require this knowledge to be configured at each node.

Compliant. The mechanism adds information to the existing Diameter capabilities exchange mechanism for determining peer support and overload control characteristics. Since this is accomplished dynamically at the start of connections, no provisioning is required to establish which peers support the mechanism and in what fashion. Implementations are free to add configuration for local policy and other control of the mechanism, but this is not required.

REQ 7: The overload control mechanism and any associated default algorithm(s) MUST ensure that the system remains stable. At some point after an overload condition has ended, the mechanism MUST enable capacity to stabilize and become equal to what it would be in the absence of an overload condition. Note that this also requires that the mechanism MUST allow nodes to shed load without introducing non converging oscillations during or after an overload condition.

Compliant. It is possible for an implementation using this to meet this requirement, and the hop-by-hop nature limits the impact of overload control actions. Additional guidance is provided for implementors on sending of the Overload-Metric and its implications for the closed loop control system created by this mechanism.

REQ 8: Supporting nodes MUST be able to distinguish current overload information from stale information, and SHOULD make decisions using the most currently available information.

Compliant. The mechanism provides for rapid updates of overload control information as well as having timeouts on the validity of overload information that must be provided by senders.

REQ 9: The mechanism MUST function across fully loaded as well as quiescent transport connections. This is partially derived from the requirement for stability in REQ 7.

Compliant. The mechanism uses piggybacked information transfer, which will generally result in the ability to transfer information on a similar rate to loading. It also provides for triggering the use of DWR with piggybacked information for quiescent connections.

REQ 10: Consumers of overload information MUST be able to determine when the overload condition improves or ends.

Compliant. The mechanism provides for rapid updates of overload control information, including abatement information, as well as mandatory timeouts on the validity of overload information that must be provided by senders (it is soft state). Additionally, the mechanism provides for sending a DWR with piggybacked information to inform of overload abatement more quickly.

REQ 11: The overload control mechanism MUST be able to operate in networks of different sizes.

Compliant. The hop-by-hop nature of the mechanism restricts the impacts that large networks might have on the ability of nodes to deal with overload control information, as well as restricting the signaling needed to convey overload information. The use of piggybacked information transfer limits the additional messaging imposed by the mechanism for large and small networks and has the characteristic of scaling with the amount of Diameter traffic on a network. Additionally, the dynamic nature of the capabilities exchange reduces the provisioning burden that can be incurred at large scales.

REQ 12: When a single network node fails, goes into overload, or suffers from reduced processing capacity, the mechanism MUST make it possible to limit the impact of this on other nodes in the network. This helps to prevent a small-scale failure from becoming a widespread outage.

Compliant. The mechanism provides for information about such issues to be conveyed in order for nodes to take appropriate action to mitigate the situation and prevent cascades.

REQ 13: The mechanism MUST NOT introduce substantial additional work for node in an overloaded state. For example, a requirement for an overloaded node to send overload information every time it received a new request would introduce substantial work. Existing messaging is likely to have the characteristic of increasing as an overload condition approaches, allowing for the possibility of increased feedback for information piggybacked on it.

Compliant. The mechanism requires sending load and overload information on all messages exchanged with compliant peers. It does not, however, require that the information be recalculated or updated with each message. The update frequency is up to the implementation, and each implementation can make decisions on balancing the update of overload information along with its other priorities. It is expected that using a periodically updated grouped AVP added to all messages sent to compliant peers will not add substantial additional work. Piggyback base transport also does not require composition, sending, or

parsing of new Diameter messages for the purpose of conveying overload control information.

REQ 14: Some scenarios that result in overload involve a rapid increase of traffic with little time between normal levels and overload inducing levels. The mechanism SHOULD provide for rapid feedback when traffic levels increase.

Compliant. The use of piggybacked information transport by the mechanism allows for overload control information to be sent at the same rate as the normal traffic. It is presumed that the rate of normal traffic will go up as nodes approach, or enter, overload. Additionally, DWR messages may be proactively triggered with piggybacked overload control information to provide overload control information transfer in an ad hoc fashion.

REQ 15: The mechanism MUST NOT interfere with the congestion control mechanisms of underlying transport protocols. For example, a mechanism that opened additional TCP connections when the network is congested would reduce the effectiveness of the underlying congestion control mechanisms.

Compliant. The mechanism does not require interaction with any underlying congestion control. It relies solely on piggybacked transport and does not request or recommend changes in how the underlying connections are performed.

REQ 16: The overload control mechanism is likely to be deployed incrementally. The mechanism MUST support a mixed environment where some, but not all, nodes implement it.

Compliant. The mechanism specifies behavior for dealing with non-supporting elements.

REQ 17: In a mixed environment with nodes that support the overload control mechanism and that do not, the mechanism MUST result in at least as much useful throughput as would have resulted if the mechanism were not present. It SHOULD result in less severe congestion in this environment.

Compliant. When dealing with supporting, and non-supporting nodes, the mechanism specifies behavior that attempts to apply relevant information to decisions on sending to non-compliant hosts. This behavior should result in reductions in traffic that increase the

likelihood of successful overload mitigation in mixed networks.

REQ 18: In a mixed environment of nodes that support the overload control mechanism and that do not, the mechanism MUST NOT preclude elements that support overload control from treating elements that do not support overload control in a equitable fashion relative to those that do. Users and operators of nodes that do not support the mechanism MUST NOT unfairly benefit from the mechanism. The mechanism specification SHOULD provide guidance to implementors for dealing with elements not supporting overload control.

Compliant. When dealing with supporting, and non-supporting nodes, the mechanism specifies behavior that attempts to apply relevant information to decisions on sending to non-compliant hosts. This allows nodes to treat non-supporting elements in a similar, and fair, fashion relative to non-supporting elements.

REQ 19: It MUST be possible to use the mechanism between nodes in different realms and in different administrative domains.

Compliant. Scoping of overload information to realms is explicitly specified by the mechanism. There are no requirements imposed by the mechanism that would prevent overload control information from crossing between adjacent nodes that were in separate administrative domains.

REQ 20: Any explicit overload indication MUST be clearly distinguishable from other errors reported via Diameter.

Compliant. A new grouped AVP conveys all overload control information, and this is transported on existing messages that are not related to overload control. No existing Diameter error codes are used by the mechanism. One new transient error code is defined by the mechanism.

REQ 21: In cases where a network node fails, is so overloaded that it cannot process messages, or cannot communicate due to a network failure, it may not be able to provide explicit indications of the nature of the failure or its levels of congestion. The mechanism MUST result in at least as much useful throughput as would have resulted if the overload control mechanism was not in place.

Compliant. Procedures are defined cases where supporting nodes become too overloaded to send overload information. No retries or sending of additional messages are required during overload that would reduce useful throughput in these situations.

REQ 22: The mechanism MUST provide a way for a node to throttle the amount of traffic it receives from a peer node. This throttling SHOULD be graded so that it can be applied gradually as offered load increases. Overload is not a binary state; there may be degrees of overload.

Compliant. The mechanism provides a 32 bit overload severity indication. Interpretation of the value is specific to the algorithm being employed. In the case of the mandatory to implement loss algorithm, the values 0-100 are used to progressively control the amount of traffic dropped.

REQ 23: The mechanism MUST provide sufficient information to enable a load balancing node to divert messages that are rejected or otherwise throttled by an overloaded upstream node to other upstream nodes that are the most likely to have sufficient capacity to process them.

Compliant. The mechanism provides information so that a load balancing node can determine that an upstream node is in overload. Additionally, it provides load information that can be used as input for balancing decisions.

REQ 24: The mechanism MUST provide a mechanism for indicating load levels even when not in an overloaded condition, to assist nodes making decisions to prevent overload conditions from occurring.

Compliant. The mechanism provides load information in each message as well as guidelines for implementing the determination of load to be sent.

REQ 25: The base specification for the overload control mechanism SHOULD offer general guidance on which message types might be desirable to send or process over others during times of overload, based on application-specific considerations. For example, it may be more beneficial to process messages for existing sessions ahead of new sessions. Some networks may have a requirement to give priority to requests associated with emergency sessions. Any normative or otherwise

detailed definition of the relative priorities of message types during an overload condition will be the responsibility of the application specification.

Compliant. Some guidance is provided for priority selection and how to deal with different priority messages is described in an example algorithm implementation.

REQ 26: The mechanism MUST NOT prevent a node from prioritizing requests based on any local policy, so that certain requests are given preferential treatment, given additional retransmission, not throttled, or processed ahead of others.

Compliant. The mechanism does not place restrictions on how decisions are made to prioritize messages.

REQ 27: The overload control mechanism MUST NOT provide new vulnerabilities to malicious attack, or increase the severity of any existing vulnerabilities. This includes vulnerabilities to DoS and DDoS attacks as well as replay and man-in-the middle attacks. Note that the Diameter base specification [RFC6733] lacks end to end security and this must be considered.

Compliant. The hop-by-hop nature of the mechanism allows existing Diameter security mechanisms to be used for securing the connections between peers. ***Detailed analysis by persons with security expertise would be beneficial.***

REQ 28: The mechanism MUST NOT depend on being deployed in environments where all Diameter nodes are completely trusted. It SHOULD operate as effectively as possible in environments where other nodes are malicious; this includes preventing malicious nodes from obtaining more than a fair share of service. Note that this does not imply any responsibility on the mechanism to detect, or take countermeasures against, malicious nodes.

Compliant. Using a hop-by-hop mechanism limits the scope of potentially malicious information. Guidance is provided for trust, in particular relative to scopes. Additional specification around trust relationships could be useful to clarify authorization of overload control information. ***Detailed analysis by persons with security expertise would be beneficial.***

REQ 29: It MUST be possible for a supporting node to make authorization decisions about what information will be sent to peer nodes based on the identity of those nodes. This allows a domain administrator who considers the load of their nodes to be sensitive information to restrict access to that information. Of course, in such cases, there is no expectation that the overload control mechanism itself will help prevent overload from that peer node.

> Compliant. The mechanism provides guidance for authorization decisions and takes no action to restrict local policy when dealing with authorization. ***Detailed analysis by persons with security expertise would be beneficial.***

REQ 30: The mechanism MUST NOT interfere with any Diameter compliant method that a node may use to protect itself from overload from non-supporting nodes, or from denial of service attacks.

> Compliant. The mechanism allows for local policy overrides for the bulk of its behavior.

REQ 31: There are multiple situations where a Diameter node may be overloaded for some purposes but not others. For example, this can happen to an agent or server that supports multiple applications, or when a server depends on multiple external resources, some of which may become overloaded while others are fully available. The mechanism MUST allow Diameter nodes to indicate overload with sufficient granularity to allow clients to take action based on the overloaded resources without unreasonably forcing available capacity to go unused. The mechanism MUST support specification of overload information with granularities of at least "Diameter node", "realm", and "Diameter application", and MUST allow extensibility for others to be added in the future.

> Compliant. The mechanism allows for flexible specification on the scope that overload control information applies to. It also allows for additional scopes to be specified as extensions.

REQ 32: The mechanism MUST provide a method for extending the information communicated and the algorithms used for overload control.

Compliant. The mechanism allows for new algorithms to be specified as extensions. It provides an AVP for communicating overload information that can be interpreted differently by different algorithms. It also provides for extension of information transmitted.

REQ 33: The mechanism MUST provide a default algorithm that is mandatory to implement.

Compliant. The mechanism specifies the drop algorithm as mandatory to implement.

REQ 34: The mechanism SHOULD provide a method for exchanging overload and load information between elements that are connected by intermediaries that do not support the mechanism.

Not Compliant. Additional analysis is needed.

Appendix C. Extending the Overload Mechanism

This specification includes two key extension points to allow for new behaviors to be smoothly added to the mechanism in the future. The following sections discuss the means by which future documents are expected to extend the mechanism.

C.1. New Algorithms

In order to provide the ability for different means of traffic abatement in the future, this specification allows for descriptions of new traffic reduction algorithms. In general, documents that define new algorithms need to describe externally-observable node behavior in sufficient detail as to allow interoperation.

At a minimum, such description needs to include:

1. The name and IANA-registered number for negotiating the algorithm (see Section 5.3).
2. A clear description of how the Overload-Metric AVP is to be interpreted, keeping in mind that "0" is reserved to indicate that no overload condition exists.
3. An example, proof-of-concept description (preferably in pseudo-code) of how nodes can implement the algorithm.

New algorithms must be capable of working with all applications, not just a subset of applications.

It is generally expected that new algorithms will make use of the available overload control information as specified in this document. However, if additional information is needed, the Load-Info AVP allows for additional optional AVPs to be included. It is recommended that designers of any new AVPs defined for this purpose consider reusing existing AVPs first, and also design their AVPS so that they may be reused by others when possible.

C.2. New Scopes

Because it is impossible to foresee all the potential constructs that it might be useful to scope operations to for the purposes of overload, we allow for the registration of new scopes.

At a minimum, such description needs to include:

1. The name and IANA-registered number for negotiating and indicating the scope (see Section 5.4).
2. A syntax for the "Details" field of the Overload-Info-Scope AVP, preferably derived from one of the base Diameter data types.
3. An explicit and unambiguous description of how both parties to the overload control mechanism can determine which transactions correspond to the indicated scope.
4. A clear and exhaustive list that extends the one in Section 2.2, indicating exactly which combinations of scopes are allowed with the new scope. This list must take into account all of the IANA-registered scopes at the time of its publication.

It is acceptable for new scopes to be specific to constructs within one or several applications. In other words, it may be desirable to define scopes that can be applied to one kind of application while not making sense for another. Extension documents should be very clear that such is the case, however, if they choose to do so.

Appendix D. Design Rationale

The current design proposed in this document takes into account several trade-offs and requirements that may not be immediately obvious. The remainder of this appendix highlights some of the potentially more controversial and/or non-obvious of these, and attempts to explain why such decisions were made they way they were.

That said, none of the following text is intended to represent a line in the sand. All of the decisions can be revisited if necessary, especially if additional facts are brought into the analysis that change the balance of the decisions.

D.1. Piggybacking

The decision to piggyback load information on existing messages derives primarily from REQ 14 in [I-D.ietf-dime-overload-reqs]: "The mechanism SHOULD provide for increased feedback when traffic levels increase. The mechanism MUST NOT do this in such a way that it increases the number of messages while at high loads."

If we were to introduce new messaging -- say, by defining a new overload control Application -- then a node in overload would be required to generate more messages at high load in order to keep overload information in its peers up-to-date.

If further analysis determines that other factors are ultimately more important than the provisions of REQ 14, several factors would need to be considered.

First and foremost would be the prohibition, in the base Diameter specification ([RFC6733]), against adding new commands to an existing application. Specifically, section 1.3.4 stipulates: "a new Diameter application MUST be created when one or more of the following criteria are met:... A new command is used within the existing application either because an additional command is added, an existing command has been modified so that a new Command Code had to be registered, or a command has been deleted." Because of this stipulation, the addition of new command codes to existing applications would require registration of entirely new application IDs for those applications to support overload control. We consider this to be too disruptive a change to consider.

By the author’s reading, there is no provision that exempts the "Diameter Common Messages" Application (Application ID 0) from the above clauses. This effectively prohibits the additional of new messages to this Application. While it may be theoretically possible to specify behavior that hijacks the DWR/DWA watchdog messages for the purpose of overload control messaging, doing so requires a complete redefinition of their behavior and, fundamentally, their semantics. This approach seems, at first blush, to be an unacceptable change to the base Application.

The remaining approach -- defining a new application for overload control -- has some promise, if we decide not to fulfill REQ 14. It remains to be seen whether the users of the Diameter protocol, including other SDOs who define applications for Diameter, are willing to specify the use of multiple Diameter Applications for use on a single reference point.

D.2. Load AVP in All Packets

Some have questioned the currently specified behavior of message senders including a Load AVP in every message sent. This is being proposed as a potential performance enhancement, with the idea being that message recipients can save processing time by examining arbitrarily selected messages for load information, rather than looking for a Load AVP in every message that arrives. Of course, to enable this kind of sampling, the Load AVP must be guaranteed to be present; otherwise, attempts to find it will occasionally fail.

The reciprocal approach, of sending a Load AVP only when the Load has changed (or changed by more than a certain amount), requires the recipient to search through the Load-Info grouped AVP in every message received in order to determine whether a Load AVP is present.

On a cursory analysis, we determined that appending a Load AVP to each message is fundamentally a cheaper operation than traversing the contents of each Load-Info AVP to determine whether a Load AVP is present.

If a later decision is made to require examination of each message to determine whether it include a Load AVP, we may be able to obtain some efficiencies by requiring Load to be the first AVP in the Load-Info AVP.

D.3. Graceful Failure

Some commenters have raised the question of whether a node can reject an incoming connection upon recognizing that the remote node does not support the Diameter overload control mechanism. One suggestion has been to add a response code to indicate exactly such a situation.

So far, we have opted against doing so. Instead, we anticipate an incremental deployment of the overload control mechanism, which will likely consist of a mixture of nodes that support and node that do not support the mechanism. Were we to allow the rejection of connections that do not support the mechanism, we would create a situation that necessitates a "flag day," on which every Diameter node in a network is required to simultaneously, and in perfect synchronization, switch from not supporting the overload mechanism, to supporting it.

Given the operational difficulty of the foregoing, we have decided that defining a response code, even if optional, that was to be used to reject connections merely for the lack of overload control support, would form an attractive nuisance for implementors. The result could easily be a potential operational nightmare for network

operators.

Authors' Addresses

Adam Roach
Mozilla
Dallas, TX
US

Email: adam@nostrum.com

Eric McMurry
Tekelec
Dallas, TX
US

Email: emcmurry@computer.org