Diameter Maintenance and Extensions (DIME)
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

S. Donovan
Oracle
E. Noel
AT&T Labs

February 14, 2014

Diameter Overload Rate Control
draft-donovan-dime-doc-rate-control-00.txt

Abstract

This specification documents an extension to the Diameter Overload Control (DOC) base solution. This extension adds a new overload control algorithm. This algorithm allows for a server to specify a maximum rate at which Diameter requests are sent to the server.

Requirements

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 RFC 2119 [RFC2119].

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a new Diameter overload control algorithm.

The base Diameter overload specification [I-D.ietf-dime-ovli] defined the loss algorithm as the default Diameter overload control algorithm. The loss algorithm allows a server to instruct a client to reduce the amount of traffic sent to the server by throttling a percentage of requests sent to the server. While this can effectively decrease the load handled by the server, it is not always sensitive to the rate of arrival of service requests at the client. If the service requests that result in Diameter transactions increases quickly then the loss algorithm can be slow to protect the stability of servers.

Consider the case where a client is handling 100 service requests per second, where each of these service requests results in one Diameter transaction being sent to a Diameter server. If the Diameter server is approaching an overload state, or is already in an overload state, it will send a Diameter overload report requesting a percentage reduction in traffic sent. Assume for this discussion that the server requests a 10% reduction. The client will then throttle ten Diameter transactions a second, sending the remaining 90 transactions per second to the server.

Now assume that the clients service requests spikes to 1000 requests per second. The client will continue to honor the servers request to throttle 10% of the traffic. This results, in this example, in the client still sending 900 Diameter transactions per second. This spike in traffic is significantly higher than the server is expecting to handle and can result in negative impacts to the stability of the server.

The server can, of course, send another overload report requesting that the client throttle 91% of requests to get back to the desired 90 transactions per second. Once the spike has abated and the client handled service requests returns to 100 per second, this will result in just 9 transactions per second being sent to the server.

One of the benefits of a rate based algorithm is that it better handles spikes in traffic. Instead of sending a request to throttle a percentage of the traffic, the rate approach allows the server to specify the maximum number of requests per second that can be sent to the server. For instance, in this example, the server could send a rate based request specifying the maximum transactions per second to be 90. The client will send the 90 regardless of whether it is receiving 100 or 1000 service requests per second.

This document extends the base Diameter overload specification to add support for the rate based overload control algorithm.

This document draws heavily on the work in the RIA SIP Overload Control working group. The definitions of the algorithms is copied almost verbatim from the SOC document [I-D.SOC-overload-rate-control].

2. Terminology and Abbreviations

Editors note - These definitions need to be made consistent with the base Diameter overload specification.

Diameter Node

A RFC6733 Diameter Client, an RFC6733 Diameter Server, and RFC6733 agent.

Diameter Endpoint

An RFC6733 Diameter Client and RFC6733 Server.

Diameter Overload Endpoint

A Diameter node that supports the Diameter Overload extension defined in [I-D.ietf-dime-ovli].

Diameter Overload Reporting Node

A Diameter overload endpoint that sends and overload report in either a Diameter request or answer.

Diameter Overload Reacting Node

A Diameter overload endpoint that receives and acts on a Diameter overload report.

3. Capability Advertisement

Editors Note: This section depends upon the completion of the base Diameter Overload specification. As such, it cannot be complete until the data model and extension mechanism are finalized in the base DOC specification. Details for any new AVPs or modifications to existing AVPs will be added in a future version of the draft after the base DOC specification has stabilized.

This extension adds the following capabilities to the OC-Feature-Vector AVP.

OLR_RATE_ALGORITHM (0x0000000000000004)

When this flag is set by the overload control endpoint it indicates that the endpoint supports the rate overload control algorithm.

Note that Diameter nodes that support this endpoint must, by definition, support both the loss and rate based algorithms. Diameter overload reacting nodes must indicate support for both the loss and rate algorithms.

It is expected that Diameter overload reporting nodes would indicate one or the other algorithms in the OC-Feature-Vector AVP.

4. Overload Report

Editors Note: This section depends upon the completion of the base Diameter Overload specification. As such, it cannot be complete until the data model and extension mechanism are finalized in the base DOC specification. Details for any new AVPs or modifications to existing AVPs will be added in a future version of the draft after the base DOC specification has stabilized.

4.1. OC-OLR AVP

This extension defines the OC-Maximum-Rate AVP to be part of the OC-OLR AVP.

When sending an overload report for the Rate algorithm, the OC-Maximum-Rate AVP is included and the OC-Reduction-Percentage AVP is not included.

This extension makes no changes to the SequenceNumber and ValidityDuration in the OC-OLR AVP. These AVPs must also be used in rate overload reports.

This extension does not define new overload report types. The existing report types of host and realm defined in [I-D.ietf-dime-ovli] apply to the rate control algorithm. The peer report time defined in [I-D.donovan-agent-overload] also applies to the rate control algorithm.

4.2. OC-Maximum-Rate AVP

The OC-Maximum-Rate AVP (AVP code TBD8) is type of Unsigned32 and describes the maximum rate that that the sender is requested to send traffic. This is specified in terms of requests per second.

Editor's note: Need to specify a maximum value.

A value of zero indicates that no traffic is to be sent.

5. Rate Based Control Algorithm

This section is pulled from [I-D.SOC-overload-rate-control], with minor changes needed to make it apply to the Diameter protocol.

5.1. Overview

The server is the one protected by the overload control algorithm defined here. This is also referred to as the reporting node. The client is the one that throttles traffic towards the server. This is also referred to as the reacting node.

Following the procedures defined in [draft-ietf-dime-doic], the server and clients signal one another support for rate-based overload control.

Then periodically, the server relies on internal measurements (e.g. CPU utilization, queueing delay...) to evaluate its overload state and estimate a target Diameter request rate in number of requests per second (as opposed to target percent reduction in the case of loss-based abatement).

When in an overloaded state, the reporting node uses the OC-OLR AVP to inform reacting nodes of its overload state and of the target Diameter request rate.

Upon receiving the overload report with a target Diameter request rate, each reacting node throttles new Diameter requests towards the reporting node.

5.2. Reporting Node Behavior

The actual algorithm used by the reporting node to determine its overload state and estimate a target Diameter request rate is beyond the scope of this document.

However, the reporting node MUST periodically evaluate its overload

state and estimate a target Diameter request rate beyond which it would become overloaded. The server must allocate a portion of the target Diameter request rate to each of its reacting nodes. The server may set the same rate for every reacting node, or may set different rates for different reacting node.

The max rate determined by the reporting node for a reacting node applies to the entire stream of Diameter requests, even though throttling may only affect a particular subset of the requests, since the reacting node might can apply priority as part of its decision of which requests to throttle.

When setting the maximum rate for a particular reacting node, the reporting node may need take into account the workload (e.g. cpu load per request) of the distribution of message types from that reacting node. Furthermore, because the reacting node may prioritize the specific types of messages it sends while under overload restriction, this distribution of message types may be different from the message distribution for that reacting node under non-overload conditions (e.g., either higher or lower cpu load).

Note that the AVP for the rate algorithm is an upper bound (in request messages per second) on the traffic sent by the reacting node to the reporting node. The reacting node may send traffic at a rate significantly lower than the upper bound, for a variety of reasons.

In other words, when multiple reacting nodes are being controlled by an overloaded reporting node, at any given time some reacting nodes may receive requests at a rate below its target Diameter request rate while others above that target rate. But the resulting request rate presented to the overloaded reporting node will converge towards the target Diameter request rate.

Upon detection of overload, and the determination to invoke overload controls, the reporting node MUST follow the specifications in [draft-ietf-dime-ovli] to notify its clients of the allocated target Diameter request rate.

The reporting node MUST use the OC-Maximum-Rate AVP defined in this specification to communicate a target Diameter request rate to each of its clients.

5.3. Reacting Node Behavior

5.3.1. Default algorithm

In determining whether or not to transmit a specific message, the reacting node may use any algorithm that limits the message rate to

1/T messages per second. It may be strictly deterministic, or it may be probabilistic. It may, or may not, have a tolerance factor, to allow for short bursts, as long as the long term rate remains below 1/T. The algorithm may have provisions for prioritizing traffic.

If the algorithm requires other parameters (in addition to "T", which is 1/OC-Maximum-Rate), they may be set autonomously by the client, or they may be negotiated independently between client and server.

In either case, the coordination is out of scope for this document. The default algorithms presented here (one without provisions for prioritizing traffic, one with) are only examples. Other algorithms that forward messages in conformance with the upper bound of 1/T messages per second may be used.

To throttle new Diameter requests at the rate specified in the OC-Maximum-Rate AVP value sent by the reporting node to its reacting nodes, the reacting node MAY use the proposed default algorithm for rate-based control or any other equivalent algorithm.

The default Leaky Bucket algorithm presented here is based on [ITU-T Rec. I.371] Appendix A.2. The algorithm makes it possible for clients to deliver Diameter requests at a rate specified in the OC-Maximum-Rate value with tolerance parameter TAU (preferably configurable).

Conceptually, the Leaky Bucket algorithm can be viewed as a finite capacity bucket whose real-valued content drains out at a continuous rate of 1 unit of content per time unit and whose content increases by the increment T for each forwarded Diameter request. T is computed as the inverse of the rate specified in the OC-Maximum-Rate AVP value, namely T = 1 / OC-Maximum-Rate.

Note that when the OC-Maximum-Rate value is 0 with a non-zero OC-Validity-Duration, then the reacting node should reject 100% of Diameter requests destined to the overloaded reporting node. However, when the OC-Validity-Duration value is 0, the client should stop throttling.

If, at a new Diameter request arrival, the content of the bucket is less than or equal to the limit value TAU, then the Diameter request is forwarded to the server; otherwise, the Diameter request is rejected.

Note that the capacity of the bucket (the upper bound of the counter) is (T + TAU).

The tolerance parameter TAU determines how close the long-term

admitted rate is to an ideal control that would admit all Diameter requests for arrival rates less than 1/T and then admit Diameter requests precisely at the rate of 1/T for arrival rates above 1/T. In particular at mean arrival rates close to 1/T, it determines the tolerance to deviation of the inter-arrival time from T (the larger TAU the more tolerance to deviations from the inter-departure interval T).

This deviation from the inter-departure interval influences the admitted rate burstyness, or the number of consecutive Diameter requests forwarded to the reporting node (burst size proportional to TAU over the difference between 1/T and the arrival rate).

Reporting nodes with a very large number of clients, each with a relatively small arrival rate, will generally benefit from a smaller value for TAU in order to limit queuing (and hence response times) at the reporting node when subjected to a sudden surge of traffic from all reacting nodes. Conversely, a reporting node with a relatively small number of reacting nodes, each with proportionally larger arrival rate, will benefit from a larger value of TAU.

Once the control has been activated, at the arrival time of the k-th new Diameter request, ta(k), the content of the bucket is provisionally updated to the value

X' = X - (ta(k) - LCT)

where X is the value of the leaky bucket counter after arrival of the last forwarded Diameter request, and LCT is the time at which the last Diameter request was forwarded.

If X' is less than or equal to the limit value TAU, then the new Diameter request is forwarded and the leaky bucket counter X is set to X' (or to 0 if X' is negative) plus the increment T, and LCT is set to the current time ta(k). If X' is greater than the limit value TAU, then the new Diameter request is rejected and the values of X and LCT are unchanged.

When the first response from the reporting node has been received indicating control activation (OC-Validity-Duration>0), LCT is set to the time of activation, and the leaky bucket counter is initialized to the parameter TAU0 (preferably configurable) which is 0 or larger but less than or equal to TAU.

TAU can assume any positive real number value and is not necessarily bounded by T.

TAU=4*T is a reasonable compromise between burst size and throttled

rate adaptation at low offered rate.

Note that specification of a value for TAU, and any communication or coordination between servers, is beyond the scope of this document.

A reference algorithm is shown below.

No priority case:

```
// T: inter-transmission interval, set to 1 / OC-Maximum-Rate
// TAU: tolerance parameter
// ta: arrival time of the most recent arrival
// LCT: arrival time of last SIP request that was sent to the server
//      (initialized to the first arrival time)
// X: current value of the leaky bucket counter (initialized to
//    TAU0)

// After most recent arrival, calculate auxiliary variable Xp
Xp = X - (ta - LCT);

if (Xp <= TAU) {
  // Transmit SIP request
  // Update X and LCT
  X = max (0, Xp) + T;
  LCT = ta;
} else {
  // Reject SIP request
  // Do not update X and LCT
}
```

5.3.2. Priority treatment

The reacting node is responsible for applying message priority and for maintaining two categories of requests: Request candidates for reduction, requests not subject to reduction (except under extenuating circumstances when there aren't any messages in the first category that can be reduced).

Accordingly, the proposed Leaky bucket implementation is modified to support priority using two thresholds for Diameter requests in the set of request candidates for reduction. With two priorities, the proposed Leaky bucket requires two thresholds TAU1 < TAU2:

- o All new requests would be admitted when the leaky bucket counter is at or below TAU1,

- Only higher priority requests would be admitted when the leaky bucket counter is between TAU1 and TAU2,

- All requests would be rejected when the bucket counter is above TAU2.

This can be generalized to n priorities using n thresholds for n>2 in the obvious way.

With a priority scheme that relies on two tolerance parameters (TAU2 influences the priority traffic, TAU1 influences the non-priority traffic), always set TAU1 <= TAU2 (TAU is replaced by TAU1 and TAU2). Setting both tolerance parameters to the same value is equivalent to having no priority. TAU1 influences the admitted rate the same way as TAU does when no priority is set. And the larger the difference between TAU1 and TAU2, the closer the control is to strict priority queueing.

TAU1 and TAU2 can assume any positive real number value and is not necessarily bounded by T.

Reasonable values for TAU0, TAU1 & TAU2 are: TAU0 = 0, TAU1 = 1/2 * TAU2 and TAU2 = 10 * T.

Note that specification of a value for TAU1 and TAU2, and any communication or coordination between servers, is beyond the scope of this document.

A reference algorithm is shown below.

Priority case:

```
    // T: inter-transmission interval, set to 1 / OC-Maximum-Rate
    // TAU1: tolerance parameter of no priority SIP requests
    // TAU2: tolerance parameter of priority SIP requests
    // ta: arrival time of the most recent arrival
    // LCT: arrival time of last SIP request that was sent to the server
    //      (initialized to the first arrival time)
    // X: current value of the leaky bucket counter (initialized to
    //    TAU0)

    // After most recent arrival, calculate auxiliary variable Xp
    Xp = X - (ta - LCT);

    if (AnyRequestReceived && Xp <= TAU1) || (PriorityRequestReceived &&
    Xp <= TAU2 && Xp > TAU1) {
      // Transmit SIP request
      // Update X and LCT
      X = max (0, Xp) + T;
      LCT = ta;
    } else {
      // Reject SIP request
      // Do not update X and LCT
    }
```

5.3.3. **Optional enhancement: avoidance of resonance**

As the number of reacting node sources of traffic increases and the throughput of the reporting node decreases, the maximum rate admitted by each reacting node needs to decrease, and therefore the value of T becomes larger. Under some circumstances, e.g. if the traffic arises very quickly simultaneously at many sources, the occupancies of each bucket can become synchronized, resulting in the admissions from each source being close in time and batched or very 'peaky' arrivals at the reporting node, which not only gives rise to control instability, but also very poor delays and even lost messages. An appropriate term for this is 'resonance' [Erramilli].

If the network topology is such that this can occur, then a simple way to avoid this is to randomize the bucket occupancy at two appropriate points: At the activation of control, and whenever the bucket empties, as follows.

After updating the value of the leaky bucket to X', generate a value u as follows:

if X' > 0, then u=0

else if X' <= 0 then uniformly distributed between -1/2 and +1/2

Then (only) if the arrival is admitted, increase the bucket by an amount T + uT, which will therefore be just T if the bucket hadn't emptied, or lie between T/2 and 3T/2 if it had.

This randomization should also be done when control is activated, i.e. instead of simply initializing the leaky bucket counter to TAU0, initialize it to TAU0 + uT, where u is uniformly distributed as above. Since activation would have been a result of response to a request sent by the reacting node, the second term in this expression can be interpreted as being the bucket increment following that admission.

This method has the following characteristics:

- If TAU0 is chosen to be equal to TAU and all sources were to activate control at the same time due to an extremely high request rate, then the time until the first request admitted by each client would be uniformly distributed over [0,T];

- The maximum occupancy is TAU + (3/2)T, rather than TAU + T without randomization;

- For the special case of 'classic gapping' where TAU=0, then the minimum time between admissions is uniformly distributed over [T/2, 3T/2], and the mean time between admissions is the same, i.e. T+1/R where R is the request arrival rate;

- At high load randomization rarely occurs, so there is no loss of precision of the admitted rate, even though the randomized 'phasing' of the buckets remains.

6. IANA Consideration

TBD

7. Security Considerations

Agent overload is an extension to the based Diameter overload mechanism. As such, all of the security considerations outlined in [I-D.ietf-dime-ovli] apply to the agent overload scenarios.

8. Acknowledgements

9. References

9.1. Normative References

[I-D.SOC-overload-rate-control]
Noel, E., "SIP Overload Rate Control", February 2014.

[I-D.ietf-dime-ovli]
Korhonen, J., "Diameter Overload Indication Conveyance", October 2013.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

9.2. Informative References

[I-D.donovan-agent-overload]
Donovan, S., "Diameter Agent Overload", February 2014.

Authors' Addresses

Steve Donovan
Oracle
17210 Campbell Road
Dallas, Texas 75254
United States

Email: srdonovan@usdonovans.com

Eric Noel
AT&T Labs
200s Laurel Avenue
Middletown, NJ 07747
United States