

RADEXT Working Group
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

A. DeKok
InkBridge Networks
20 October 2025

Proxy Load Balancing in the Remote Authentication Dial In User Service
(RADIUS) Protocol
draft-dekok-radext-proxy-load-00

Abstract

This document describes a few methods which can assist operators of proxies in the in the Remote Authentication Dial In User Service (RADIUS) Protocol. The methods described here improve proxy load balancing and rate limiting, without requiring any changes to the underlying protocol. While these methods have been shown to work in practice, there has been insufficient experience with them to publish this document either as standards track, or as a best current practice.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at
<https://datatracker.ietf.org/doc/draft-dekok-radext-proxy-load/>.

Discussion of this document takes place on the RADEXT Working Group mailing list (<mailto:radext@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/radext/>. Subscribe at <https://www.ietf.org/mailman/listinfo/radext/>.

Source for this draft and an issue tracker can be found at
<https://github.com/freeradius/proxy-bcp.git>.

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 <https://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 23 April 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Deterministic Load Balancing to Home Servers	3
3.1. Problem	4
3.2. Solution	4
4. Rate Limiting High Volume Access-Requests	5
4.1. Problem	5
4.2. Existing Solutions	6
4.3. Rate Limiting	6
4.3.1. Utility	7
4.3.2. Algorithm	7
4.3.3. Cache Configuration Parameters	9
4.3.4. Cache Entry Parameters	12
4.4. Delay instead of Rate Limit	14
4.5. Inappropriate Use Cases	15
5. IANA Considerations	15
6. Privacy Considerations	15
7. Security Considerations	16
8. Acknowledgements	16
9. References	16
9.1. Normative References	16
9.2. Informative References	16
Author's Address	17

1. Introduction

The Remote Authentication Dial In User Service (RADIUS) protocol defines proxying in [RFC2865], Section 2.3. As discussed in PROXY-BCP, there has been little progress in defining load-balancing methods for proxying. This document describes a number of methods which have been shown to work, and do not require protocol changes.

However, there has been insufficient experience with them to publish this document either as standards track, or as a best current practice.

The following sections outline the underlying problems, and then the specific methods which can be used to address those problems.

2. Terminology

Proxy

A RADIUS proxy as per [RFC2865].

Unless specifically mentioned, any discussion of proxying is assumed to apply to all possible RADIUS packet Codes.

Similarly, unless specifically mentioned, any discussion of proxying is assumed to apply to all possible RADIUS transport protocols.

Home Server

A RADIUS server which authenticates the user, or performs the final storage of accounting information. i.e. The authoritative server which does not proxy packets to any other destination.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Deterministic Load Balancing to Home Servers

The RADIUS protocol does not define a way to deterministically load balance packets across a set of Home Servers. This section describes a method which can do that, even when the set of active Home Servers changes dynamically.

The method outlined here is suitable only when a server is sending packets to one or more Home Servers. It MUST NOT be used then the next hop is another proxy. The method outlined here can also be used by RADIUS clients.

The method outlined here is also suitable only when the Access-Request packets contain a State attribute, such as with EAP or any challenge-response mechanism.

3.1. Problem

In some cases, authentication requires one or more round trips of Access-Request / Access-Challenge, before any Access-Accept or Access-Reject is seen. The challenge in a proxy environment is to deterministically send all packets for one session to the same Home Server. When packets are for one session are sent to different Home Servers, authentication is unlikely to succeed.

The above requirement stands in conflict with the need to load-balance requests across Home Servers. While this load balancing can be done via hashing the contents of User-Name or Calling-Station-Id, there is a problem when set of Home Servers changes over time.

This problem can largely be addressed by using consistent hashing for Home Servers, but a naive consistent hashing is unaware of RADIUS protocol issues. The result of any consistent hashing will still result in packets for one authentication session being sent to different Home Servers, where authentication will fail.

3.2. Solution

TBD - full details.

- * Each Home Server has a statically configured and unique 8-bit identifier
- * On receiving the initial Access-Request (without State), the proxy does load balancing via some method.
- * On receiving the Access-Challenge with a State attribute, it increases the size of the State attribute by one octet, and adds the unique Home Server identifier. It then returns the response as normal
- * On receiving an Access-Request with State, it removes the Home Server Identifier, and then sends the modified request to the Home Server.

- * If the Home Server is down, then depending on the configuration, the proxy either discards the packet, or sends a synthesized Access-Reject, or sends a Protocol-Error.

This method does not offer any different experience for the supplicant or user being authenticated. There is no magic way to force authentication to succeed when the relevant Home Server is down.

However, this method can decrease the load on the Home Servers substantially when a failure occurs. i.e. if there are 4 home servers and one goes down, the load on the remaining ones will increase by 25%, all of which is useless traffic. If there are two home servers, then the load on the remaining one will double, at least temporarily.

The cost here is a small amount of configuration, and a tiny amount of extra work on the proxy.

The benefit of this method is that less useless work is done, and better decisions are made based on actual network status, instead of on ad hoc guesses. The result is a network which is more stable.

4. Rate Limiting High Volume Access-Requests

Existing standards suggest that RADIUS clients rate limit their requests ([RFC5080], Section 2.2.2). Similarly, (802.1X) suggests that supplicants and authenticators rate-limit the packets that they send. Unfortunately, experience shows that rate limiting is often not performed.

4.1. Problem

In practice, operators see one of two behaviors with badly behaved supplicants and clients. Either the client sends thousands of Access-Requests a second without waiting for a response, or the client sends an Access-Request immediately after receiving an Access-Reject. These packets are nearly always for one particular user session, and other user sessions on the same client generally behave correctly.

This high rate of packets causes unnecessary load on RADIUS servers, proxies, and proxy chains such as [EDUROAM] and [OPENROAMING]. In some cases, these problematic packets will be the majority of packets seen by a RADIUS server.

The cost of managing these packets is not small. Proxies can apply complex policies to each packet, which is expensive. Home Servers can apply complex policies, including multiple database lookups.

As such, these packets increase the load on the entire RADIUS infrastructure for no possible benefit. It is therefore useful to detect and block these packets, using a method which is both cheap and has minimal impact on other users / packets / sessions which are behaving properly.

4.2. Existing Solutions

The solution until now has been to simply block the offending RADIUS client completely. While this step protects the server which is receiving those packets, it has unfortunate side effects.

The client being blocked may be an Access Point (AP) which is managing other user sessions. Or, the client being blocked may be a RADIUS proxy. In either case, this blocking will negatively effect users are are behaving properly.

In addition, this blocking is generally added manually, and removed manually. This manual blocking is expensive and slow. It relies on people to notice problemn situations, and take action on human time frames

In some cases it is done automatically, based on packet rates. This automatic blocking cannot generally distinguish between high rates of normal traffic due to an unusual situation (e.g. power outage and restoral), or high rates due to one misbehaving supplicant or client.

The proposal outlined below solves these issues.

4.3. Rate Limiting

The rate limiting method proposed here depends on a few key pieces of information, and being performed at particular stages of a server which processes packets. This server is typically a proxy, but it could also be a Home Server.

The discussion below focusses on Access-Request packets, and Access-Reject responses. However, similar rate limiting could be performed for Access-Accept responses, or Accounting-Request packets. Since the main problem seen in practice is with Access-Request / Access-Reject, those packets are what we focus on here.

4.3.1. Utility

This method does not require protocol changes, and therefore does not require coordination with other clients or servers.

This method also has no negative interactions with itself. Multiple proxies can implement the same method, and there will be no negative affects. In short, if any one proxy implements this method, then all "next hop" proxies and home servers are protected, whether or not they implement this method.

This method also has no effect on well-behaved sessions. Sessions which behave normally do not trigger the rate limiting method. Sessions which are rejected and then rate-limit their own future requests are also unaffected.

The main cost to this method is one lookup for each Access-Request packet received by a server. Where the packet will not be rate-limited, no further action is taken. Additional work is only performed if the the Access-Request packet is to be rate limited, or where the server is processing an Access-Reject.

4.3.2. Algorithm

The rate limiting method is essentially a negative cache, with extensions that allow it to be applied to RADIUS; to ensure that the cache does not negatively affect normal traffic; and to allow it to be managed within a RADIUS environment.

The rate limit algorithm is performed as follows, via the pseudo-code outlined below:

When receiving an Access-Request, look up the packet in the cache based on a Key (see below). The Key is typically either Calling-Station-ID, User-Name, or some combination thereof.

If there is no cache entry for that key, the server continues processing the packet as normal. No further changes to the cache are made.

If the cache entry indicates that rate limiting is not being performed, the server continues processing the packet as normal. No further changes to the cache are made.

Otherwise, there is a cache entry for that key, and rate limiting is being performed for it. The system updates timers and counters associated with the entry, and return the cached response. This response is typically an Access-Reject. No further processing of the packet is performed.

When receiving an Access-Reject, either create or update an entry in the cache, based on the key which is taken from the Access-Request.

When creating a cache entry, the cache entry contains both the Access-Reject, along with various counters and timers which are discussed below. The initial state of the cache entry is set to not rate limit input packets. The response is then processed as normal through the server, typically resulting in the response being sent to the RADIUS client.

When updating a cache entry, the counters and timers are updated. If a configurable threshold is reached, then the cache entry is set enable rate limiting. The response is then processed as normal through the server, typically resulting in the response being sent to the RADIUS client.

Note that these checks are performed only for the initial Access-Reject response. If the server receives identical retransmissions of one Access-Reject response, then those retransmissions do not affect the cache in any way.

As an extension to the above algorithm, the cache can also track the number of requests which have been sent before any response is seen. If many requests for the same Key are seen in a short time window, then rate limiting can also be enabled, even before a response is seen. The cache parameters defined below provide for this capability, even though the outline of the algorithm above does not discuss it.

The system also tracks additional information which enables it to manage the cache. This management includes limiting the total size of the cache, expiring old cache entries which have not been used in a while, and any other necessary bookkeeping.

The result of this process is minimal disruption to normal and well-behaved traffic, while quickly blocking problematic sessions.

The only real limitation of this method is that it depends on a fixed key. That is, it requires that misbehaving supplicants or clients are sending essentially identical identifying information in each packet. In practice, this behavior is what is seen.

This cache does not protect against malicious attackers who either gain trusted access to the RADIUS network, or from attackers who send enormous amounts of requests via an untrusted supplicant. Protection against those attacks is a separate subject, and we do not further address them here.

4.3.3. Cache Configuration Parameters

This section outlines the configuration parameters which can be used to control both the cache as a whole. It is possible to use other parameters but those are not described here.

The cache can also track any statistics which are useful to the local administrator. Common statistics may include maximum cache usage, packets per second, cache hit rate, cache miss rate, etc.

Cache implementations should also provide for interfaces to examine the cache and to create, delete, or update individual entries.

The cache as a whole has the following configuration parameters:

Maximum Size

This parameter is an integer value which limits the total number of entries.

As the number of misbehaving sessions is typically tiny (and typically one), this configuration parameter exists only to prevent the cache itself from being attacked.

The default Maximum Size value should generally be a small number, especially when Threshold (see below) is set to one (1). Since the cache is expected to have only one entry in the worst case, the default values for Maximum Size can be kept low.

Since each cache entry is small, there are minimal problems with setting the Maximum Size to larger values. The system may use more memory, with minimal other side effects.

However, if Threshold is set to a larger value (e.g. 2 or 3), then the cache may have many entries, even though only a small number of cache entries are being actively rate limited. In that situation, it is likely useful to increase the Maximum Size value to one which allows a large number of cache entries to be maintained.

That number should be large enough to create cache entries for all possible Access-Rejects that are seen within the Window time period.

Key

This parameter is a field which contains the key used to uniquely identify and index each cache entry.

The default Key value should be a combination of Calling-Station-Id, User-Name, and possible Called-Station-Id. That combination ensures that the cache entries are scoped to the smallest possible source.

Initial Lifetime

This parameter is a time interval which sets the default lifetime of a cache entry.

When a cache entry is created, it has a Lifetime (see below) which is set to this Default Lifetime. If no further updates to the cache are performed, the cache entry expires at its Lifetime, and is deleted.

The default Initial Lifetime value should be between one (1) and five (5) seconds. Using a "too low" value means that the system is less likely to detect problematic packets. Using a "too high" value will not benefit the system in any way, but will also have little impact due to the small number of entries in the cache.

Extension Time

This parameter is a time interval which extends the cache Lifetime when an entry is updated.

Note that updates to the cache entries should also be rate limited, typically to once per second. Otherwise, the system could receive 1000 packets each second, and then also update the cache entry Lifetime 1000 times in that same second.

Alternatively, the configuration could be simplified by using the same parameter for both Initial Lifetime and Extension Time. The parameters are described separately here just to make it clear that there may be some benefit to making them different.

The default Extension Time value should be between one (1) and five (5) seconds. Using a "too low" value means that many cache updates are performed for no additional benefit. Using a "too

high" value will not benefit the system in any way, but will also have little impact due to the small number of entries in the cache.

Request Window

This parameter is a time interval which is used to detect misbehaving sessions before any response has been received.

Once a cache entry is created, it is set to not perform rate limiting. However, if the cache sees another Access-Request packet using the same Key during the Request Window time interval, then rate limiting may be enabled for that cache entry.

The default Request Window values should be between one (1) second and five (5) seconds. Values much higher than that are likely to be unhelpful, as few systems will be overloaded with one problematic packet every 5-30 seconds.

Request Treshhold

This parameter is an integer which is used to initiate rate limiting for a cache entry before any response has been received.

Rate limiting is enabled when the cache sees "Request Threshold" different Access-Request packets within the "Request Window" time period. Note that retransmitted packets are not counted as different packets for the purposes of this counter.

This parameter is here to allow better management of the cache, if necessary. In most situations, setting Request Threshold to one (1) is sufficient. Request Threshold values higher than that are less likely to be useful, as they will minimize the utility of the cache.

Response Window

This parameter is a time interval which is used to detect misbehaving sessions which receive Access-Rejects.

Once a cache entry is created, it is set to not perform rate limiting. However, if the cache sees another Access-Reject packet using the same Key during this Window time interval, then rate limiting may be enabled for that cache entry.

The default Response Window values should be between one (1) second and five (5) seconds. Values much higher than that are likely to be unhelpful, as few systems will be overloaded with one problematic packet every 5-30 seconds.

Response Treshhold

This parameter is an integer which is used to initiate rate limiting for a cache entry.

Rate limiting is enabled when the cache sees "Response Threshold" different Access-Reject packets within the "Response Window" time period. Note that as discussed earlier, responses to retransmitted packets are not counted as different packets.

This parameter is here to allow better management of the cache, if necessary. In most situations, setting Response Threshold to one (1) is sufficient. Response Threshold values higher than that are less likely to be useful, as they will minimize the utility of the cache.

Where the proxy chain experiences a high volume of rejects in normal situations, the Response Threshold value SHOULD be set to value higher than one (1), but which is still small. It is difficult to make a more specific recommendation, as the distribution of traffic in each network can vary significantly.

If the proxy chain experiences a high volume of rejects in normal situations, the cache will contain a large number of cache entries, even though only a small number may be actively rate limited.

4.3.4. Cache Entry Parameters

This section outlines the configuration parameters which can be used to control both the cache as a whole. It is possible to use other parameters but those are not described here.

The cache entry can also track any statistics which are useful to the local administrator. Common statistics may include the time at which the entry was created, the time at which it was last updated, the time at which it was last used, number of Access-Request packets which match the Key, along with any other information which may be useful.

Each cache entry tracks the following parameters:

Key

The data used to find or index this cache entry. This field corresponds to the Key configuration for the cache.

Rate Limit

A boolean variable which indicates whether or not this cache entry will result in rate limiting input packets that match the Key.

On creation, this field is set to "false". Once set to "true", it is never changed to "false". The better way to stop rate limiting is to simply delete the cache entry.

Request Counter

An integer variable which tracks the number of different Access-Requests seen that match this Key, before any response is received.

Each time that the Request Counter field is increased to values higher than one (1), the Lifetime field (see below) should also be increased by Extension Time. This extension allows the cache to better react to abusive packets, while quickly expiring entries that may be associated with normal packets.

When the Request Counter value exceeds the configured cache Request Threshold value, the Rate Limit field is set to "true". Once the Rate Limit field is set to "true", the two Counter fields are no longer updated

Response Counter

An integer variable which tracks the number of different Access-Rejects seen that match this Key.

Each time that the Counter field is increased to values higher than one (1), the Lifetime field (see below) should also be increased by Extension Time. This extension allows the cache to better react to abusive packets, while quickly expiring entries that may be associated with normal packets.

When the Response Counter value exceeds the configured cache Threshold value, the Rate Limit field is set to "true". Once the Rate Limit field is set to "true", the two Counter fields are no longer updated

Lifetime

A time variable which indicates at which time the cache entry is deleted. As noted above, this variable can be updated when a cache entry is extended due to repeated abusive packets.

Response

A copy of the Access-Reject response packet contents.

When a particular Key is rate limited, the cache replies with a cached copy of the response. Note that this caching assumes that all responses have identical packet contents, even if the RADIUS header is different.

Due to the requirements of [RFC3579], the cached response SHOULD include an EAP-Message attribute if the Access-Request contained an EAP-Message attribute. The EAP-Message in the response MUST be an EAP Failure, and MUST be updated to include the same EAP Identifier field as is seen in the current Access-Request.

4.4. Delay instead of Rate Limit

Instead of rate limiting requests, the cache could be used to delay them. This approach is less rigid than rate limiting, but still achieves many of the same results. This possibility is noted separately from rate limiting, because of the impact it has on proxy chains.

When packets are rate limited due to abuse, that rate limiting is "safe", in that multiple servers in a proxy chain can each apply the rate limiting, and there will be no negative interactions. In contrast, if a server delays a packet, those delays may add up across a proxy chain, and cause negative outcomes.

As such, only the first server in a proxy chain can delay requests. That is when the client is known to be a NAS, delaying requests can be useful. When the client is known to be another proxy server, the receiving server MUST NOT delay requests.

When a server suspects that abuse is occurring, it can choose to delay requests for a short period of time instead of forwarding them. This time period can be very short, such as 10ms, or 100ms. That is, it should be short enough to cause minimal noticeable issues for authentication, but long enough to catch the abusive packets described above.

This delay can increase the likelihood of the server catching abusive packets, as noted above.

4.5. Inappropriate Use Cases

This method is intended to protect RADIUS systems from short bursts of abusive traffic from a single source, which is sent at high rates. As such, it is NOT RECOMMENDED for other uses. We describe some other possible use cases here, and explain why this solution is not appropriate for them.

One issue which may seem useful is to address the problem of a large volumes of otherwise normal traffic being rejected. For example, user accounts could be authentic, but could not be authorized for a particular network or location. The network will reject the request, and supplicant will see what it believes to be an erroneous reject for authentic credentials. The supplicant will then repeat its authentication attempt, generally until such time as it is outside of the problematic network.

Another issue which could be addressed is the problem of devices still having credentials for accounts which were removed a long time in the past. For example, information from the [EDUROAM] national proxies indicate that a significant percentage of traffic is Access-Rejects for accounts which have not existed for many years.

A possible solution to these issues would be to extend the Window and Default Lifetime configuration parameters to minutes or hours. This change would lower the number of repeated authentications being sent across a chain of proxies. However, these extensions can have negative side effects which are not seen in the intended use-case.

One negative side effect is simply the long lifetime of a negative cache entry. If the lifetime is a few seconds, then the network will quickly recover from Access-Rejects that are sent to temporary failures, or due to misconfigurations. Longer lifetimes are much more likely to affect users, and therefore increase the volume of support queries,

5. IANA Considerations

This document has no actions for IANA.

6. Privacy Considerations

There are no privacy considerations in this specification. It does not add or change any transport protocols, and it does not change the roles of any participant in the RADIUS ecosystem.

7. Security Considerations

This specification has no direct impact on security. It does not add or change any existing cryptography in RADIUS. It is compatible with all RADIUS standards, including (ALPN draft).

This specification does, however, recommend ways that networks can be better protected from a variety of attacks, misconfiguration, and improper implementations. As a result, RADIUS proxy networks which implement this specification should become more stable, and therefore more secure.

8. Acknowledgements

This document came out of discussions both in the RADEXT WG, and in the 2025 RADIUS Technical Conference in Tampere, Finland.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <<https://www.rfc-editor.org/rfc/rfc2865>>.
- [RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, DOI 10.17487/RFC5080, December 2007, <<https://www.rfc-editor.org/rfc/rfc5080>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

9.2. Informative References

- [EDUROAM] eduroam, "eduroam", n.d., <<https://eduroam.org>>.
- [OPENROAMING] Alliance, W. B., "OpenRoaming: One global Wi-Fi network", n.d., <<https://wballiance.com/openroaming/>>.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, DOI 10.17487/RFC3579, September 2003, <<https://www.rfc-editor.org/rfc/rfc3579>>.

Author's Address

Alan DeKok (Ed)
InkBridge Networks
Email: aland@inkbridgenetworks.com