Workgroup: Registration Protocols Extensions
Internet-Draft:
draft-loffredo-regext-epp-over-http-02
Published: 13 June 2022
Intended Status: Standards Track
Expires: 15 December 2022
Authors: M. Loffredo
IIT-CNR/Registro.it
L. Luconi Trombacchi
IIT-CNR/Registro.it
M. Martinelli
IIT-CNR/Registro.it
J. Romanowski
NASK/.pl Registry
M. Machnio
NASK/.pl Registry

Extensible Provisioning Protocol (EPP) Mapping over HTTP

Abstract

This document describes how the Extensible Provisioning Protocol (EPP) is mapped over the Hypertext Transfer Protocol (HTTP). This mapping requires the use of the Transport Layer Security (TLS) protocol to protect information exchanged between an EPP client and an EPP server.

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 15 December 2022.

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

Although the Extensible Provisioning Protocol (EPP) core specification [RFC5730] does not state the protocol used for the transit of EPP messages, only the mapping over TCP [RFC5734] has been standardized thus far. Nevertheless, some EPP implementations leverage HTTP due to its ease of use and simplicity. This document describes the reasons behind using HTTP as a substrate for EPP and how EPP is mapped over HTTP preserving the semantics of commands.

HTTP is defined in some IETF documents according to the versions currently in use: HTTP/1.1 [RFC9112], HTTP/2 [RFC9113], HTTP/3 [RFC9114]. As the differences among such versions do not affect the EPP mapping described in this document, hereinafter the version number is omitted except for presenting the special features in the underlying layers of the HTTP stack.

Stateful EPP sessions are maintained across HTTP requests through storing the state in HTTP cookies [RFC6265].

Security services beyond those defined in EPP are provided by the Transport Layer Security (TLS) protocol [RFC8446] [RFC9155].

1.1. Conventions Used in This Document

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.

2. Reasons behind Using EPP over HTTP

Many web applications and REST APIS are built on top of HTTP. This is due for a variety of reasons [RFC9205].

HTTP is loosely coupled with the network and provides client-server cross-platform technology communication. Indeed, since an HTTP connection is a higher-level abstraction of a network connection, there is no need to take over all of the lower-level details of transport protocols. For example, while in TCP the data transmission between a client and a server starts only after having established a connection through a 3-way handshake (i.e. SYN, SYN-ACK, ACK), HTTP uses a one-way communication so that a client can directly issue a request to a server and then receive a response.

Libraries and frameworks, commonly available on both client and server sides, save programmers from managing the HTTP connections. Service providers are only required to process the requests and return the responses, while consumers need only to send requests and process responses. Definitively, HTTP ease of use and simplicity reduces the development time.

Moreover, implementers can leverage the features that are available in both HTTP and its underlying layers to provide the security services needed by their applications.

With specific regard to the implementation of EPP over HTTP, some additional considerations can be made.

HTTP is stateless but not sessionless. This means that, by making an EPP session untied from the network connection, the EPP communication over HTTP is more flexible and efficient than over TCP.

The main reason supporting the usage of EPP over TCP [RFC5734] has always been its speed. TCP has been significantly faster than HTTP as HTTP was initially built on top of TCP so that every HTTP request had to be issued on a new TCP connection. However, subsequent HTTP

versions have been defined over time to increase the protocol speed and reduce the gap with transport protocols:

* Compared to the original HTTP specification, HTTP/1.1 introduced the "keep-alive" connection by default to enable a request-response sequence on a single TCP connection without repeating the connection handshake at each request;

* As opposed to HTTP/1.1, which keeps all requests and responses in plain text format, HTTP/2 defined the binary framing layer to encapsulate all messages in binary format;

* HTTP/3 is based on QUIC transport protocol [RFC9000]. QUIC uses UDP [RFC768] instead of TCP to exchange packets between the client and the server. It incorporates TLS whereas HTTP/1.1 and HTTP/2 define TLS as an add-on. So doing, HTTP/3 can provide a very quick handshake to establish a secure connection.

From the perspective of moving to the cloud to achieve scalability and cost reduction, it should be further noted that application protocols that aren't based on HTTP can be hardly migrated by using cloud-native features, on both client and server sides. In addition, from the security point of view, registries would be limited in terms of the third-party security services available to protect their EPP servers.

Finally, some considerations should be done about load balancing which is generally used by EPP operators to distribute the requests across a pool of servers and, consequently, provide an efficient domain registration and maintenance service. While HTTP load balancers are very common and are quite often software, TCP load balancers are usually implemented in dedicated hardware. In addition, HTTP load balancers don't merely forward the traffic but can make high-level routing decisions based on the message content. With regard to the performance, although HTTP load balancers do more work, their throughput is evaluated considerably fast.

Additional notes on how EPP sessions can be managed in HTTP load balancing are included in Appendix A.

3. Message Exchange

EPP describes client-server interaction as a command-response exchange where the client sends one command to the server and the server returns one response to the client. A client MUST use the POST method (Section 3.3 of [RFC7231]) to issue an EPP command through the request body. A server receiving a request MUST return an EPP message in the response body using the "Content-Length" entity-header field to indicate the length in decimal number of OCTETs of the entity-body. No EPP message information MUST be issued

through any other part of the request or the response. If the HTTP connection is closed after a server receives and successfully processes a command but before the response can be returned to the client, the server MAY attempt to undo the effects of the command to ensure a consistent state between the client and the server.

Commands MUST be processed independently and in the same order as received from the server. An EPP client MAY issue multiple EPP commands to an EPP server on an HTTP connection by relying on the HTTP keep-alive capability. A server SHOULD limit a client to a maximum number of HTTP connections based on server capabilities and operational load.

A client might be able to realize a slight performance gain by pipelining the requests, but this feature does not change the basic single command, single response operating mode of the EPP protocol. A server SHOULD limit the amount of time required for a client to issue a well-formed EPP command and, consequently close an open HTTP connection.

4. Session Management

The EPP session is implemented by using the mechanism described in [RFC6265]. An EPP session is started by the client issuing an EPP <login> command. A server receiving an EPP <login> command MUST use the "Set-Cookie" response header to send the client a token that the client will return in future requests within the scope of the EPP session. For example (Figure 1), the server can send the client a "session identifier" (a.k.a "session ID") named SID. The client then returns the session ID in the "Cookie" header of the subsequent requests.

```
== Server -> Client ==

Set-Cookie: SID=52ceb07c2a824f09a1c6f9c45574097d

== Client -> Server ==

Cookie: SID=52ceb07c2a824f09a1c6f9c45574097d
```

Figure 1

The name of the cookie attribute identifying the session ID is not relevant and depends on the implementations. Examples of the names that some programming languages use to represent the session ID include JSESSIONID (Java EE), PHPSESSID (PHP), and ASPSESSIONID (Microsoft ASP).

An EPP session is ended by the client issuing an EPP <logout> command. A server receiving an EPP <logout> command MUST end the EPP session invalidating it after having issued the <logout> response.

A client MAY open multiple EPP sessions and distribute commands from a single EPP session over multiple HTTP connections. A server SHOULD limit a client to a maximum number of EPP sessions based on server capabilities and operational load.

EPP sessions that are inactive for more than a server-defined period MAY be ended by a server invalidating the session.

Clients MAY issue the <hello> command outside an EPP session. In such a case, servers MUST return the <greeting> response without starting a session. To accomplish this, a server MAY return no cookie at all or provide the client with an expired cookie so that it cannot be used for further communication with the server. Clients MAY also issue the <hello> command within an EPP session to keep it alive.

The mechanism implemented by a server to maintain the relationship between a session and the EPP information negotiated with the client through the <login> command (e.g. the language, the namespace URIs representing both the objects and the extensions to be managed during the session) is out of the scope of this document.

The state machine described in Section 2 of [RFC5730] is updated as shown in Figure 2.

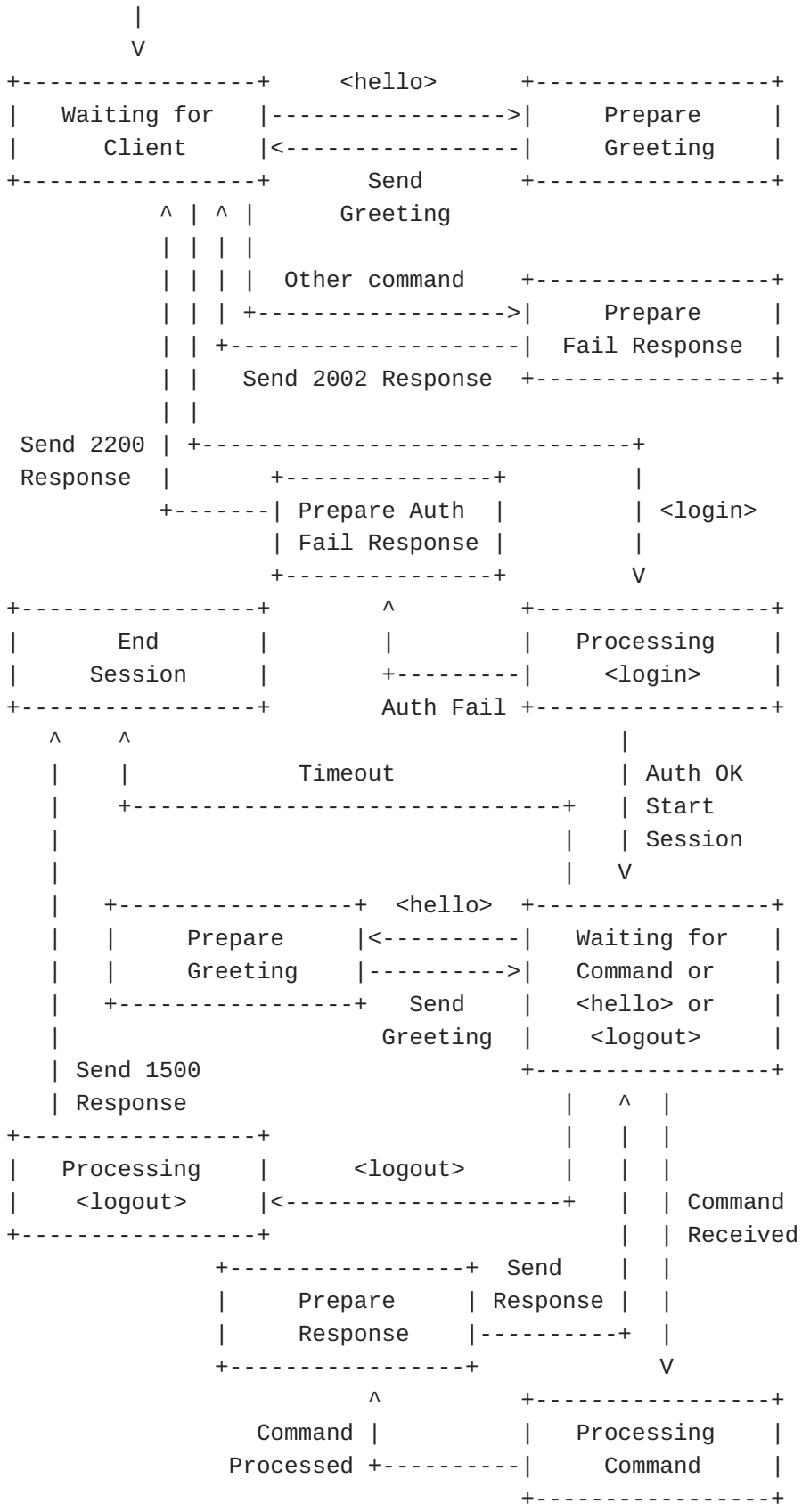
Waiting for Client
<hello>
Prepare Greeting
Send Greeting
Other command
Prepare Fail Response
Send 2002 Response
Send 2200 Response
Prepare Auth Fail Response
<login>
End Session
Auth Fail
Processing <login>
Timeout
Auth OK
Start Session
<hello>
Prepare Greeting
Send Greeting
Waiting for Command or <hello> or <logout>
Send 1500 Response
Processing <logout>
<logout>
Command Received
Send Response
Prepare Response
Command Processed
Processing Command

Figure 2

5. Return Codes

Servers MUST NOT use HTTP return codes to signal clients about the failure of the EPP commands. The HTTP code 200 MUST be used for both successful and unsuccessful EPP requests. Servers MUST use HTTP codes to signal clients about the failure of the HTTP requests.

Servers MUST return a 2002 response (i.e. Command use error) if the client issues an EPP command other than the <hello> and the <login> commands through HTTP requests including either an empty or an invalid session ID. Servers receiving a <login> command through an HTTP request including a session ID MAY return a 2002 response (i.e. Command use error) or simply ignore the incoming session ID.

6. Implementation Status

NOTE: Please remove this section and the reference to RFC 7942 prior to publication as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

6.1. IIT-CNR/Registro.it EPP Server

- Responsible Organization: Institute of Informatics and Telematics of National Research Council (IIT-CNR)/Registro.it
- Location: https://epp.nic.it/ EPP endpoint available only "per IP address" basis.
- Description: The .it EPP server is deployed on WildFly Application Server. TLS versions supported are 1.2 and 1.3. Load balancing is implemented with NGINX. EPP sessions are maintained on a Redis cluster.
- Level of Maturity: This is a live implementation.

*Coverage: This implementation includes all of the features described in this specification except for the media type that is currently set to "text/xml".
*Contact Information: Mario Loffredo, mario.loffredo@iit.cnr.it

6.2. .pl domain Registry (NASK) EPP Server

*Responsible Organization: .pl domain Registry (NASK)/dns.pl
*Location: https://dns.pl EPP endpoint available only "per IP address" basis.
*Description: It is an implementation of the EPP protocol that is used by .pl Registry.
*Level of Maturity: This is a live implementation.
*Coverage: This implementation includes all of the features described in this specification.
*Contact Information: Marcin Machnio, info@dns.pl

7. Mapping Considerations

Section 2.1 of the EPP core specification [RFC5730] describes considerations to be addressed by the transport protocol mappings. HTTP is commonly intended as a Layer 7 stateless protocol that can be used as a substrate for web applications and REST APIs. Despite those considerations have explicitly been defined for Layer 4 protocols, some of them are addressed by this document using a combination of features defined by this mapping and features provided by HTTP as follows:

*Section 3.9.3 of [RFC8095] includes features to provide reliability, flow control, ordered delivery, and congestion control of HTTP over TCP. Analogous features implemented by QUIC are described in [RFC9000].

*Section 3 and Section 4 of this document describe how the stateful nature of EPP is preserved through controlled message exchanges and managed sessions.

*Section 3 of this document notes that command pipelining is possible with HTTP, though batch-oriented processing (combining multiple EPP commands in a single HTTP request) is not permitted.

8. IANA Considerations

This document has no actions for IANA.

9. Internationalization Considerations

Servers MUST use the "charset" attribute in the HTTP "Content-Type" response header field to specify the UTF-8 character encoding (e.g. Content-Type: application/epp+xml; charset=UTF-8).

10. Security Considerations

Since clients credentials are included in the EPP <login> command, the HTTP over TLS [RFC8740] MUST be used to protect them from disclosure while in transit. As well, the transfer over TLS prevents from sniffing the session ID and, consequently, impersonating a client to perform actions on registrars' objects. Servers are REQUIRED to support TLS 1.2 [RFC8446][RFC9155] or higher.

Anyway, servers are RECOMMENDED to implement additional measures to verify the client. These measures include IP whitelisting and locking the session ID to the client's IP address.

As a further measure to enforce the security, servers SHOULD require clients to present a digital certificate. Clients who possess and present a valid X.509 digital certificate, issued by a recognized Certification Authority (CA), could be identified and authenticated by a server who trusts the corresponding CA. This certificate-based mechanism is supported by HTTPS and can be used with EPP over HTTP.

About sessions, session IDs SHOULD be randomly generated to mitigate the risk of obtaining a valid one through a brute-force search. A session ID SHOULD be at least 128 bits or 16 bytes long. An example of a reliable session ID is the Universally Unique Identifier (UUID). Servers MAY limit the lifetime of active sessions to avoid them being exchanged for a long time.

The following measures MAY also be taken to control cookies usage:

- restricting their scope through the "Domain" and "Path" attributes;
- limiting their lifetime through the "Max-Age" and "Expire" attributes.

Other attributes that are normally used to secure the cookies and prevent them to be accessed by unintended parties or scripts, such as "HttpOnly" and "Secure", are meaningless in this context.

Finally, servers are RECOMMENDED to perform additional checks to limit the rate of open EPP sessions and HTTP connections to mitigate the risk of congestion of requests. Here again, IP whitelisting could also be implemented to prevent DDoS attacks.

If the EPP server is configured as a load balancer routing the requests to a pool of backend servers, some of the aforementioned checks SHOULD be implemented on the load balancer side.

11. Acknowledgements

The authors would like to acknowledge the following individuals for their contributions to this document: Cristian Lucchesi, Stefano Ruberti, Luca Vasarelli, Roberto Ravazzolo from IIT-CNR/Registro.it and Adrian Prokop, Sławomir Mateuszczyk from NASK/.pl Registry.

12. References

12.1. 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/info/rfc2119>.

[RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, DOI 10.17487/RFC3470, January 2003, <https://www.rfc-editor.org/info/rfc3470>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <https://www.rfc-editor.org/info/rfc5730>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC6839] Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, DOI 10.17487/RFC6839, January 2013, <https://www.rfc-editor.org/info/rfc6839>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8095] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <https://www.rfc-editor.org/info/rfc8095>.

[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/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8740] Benjamin, D., "Using TLS 1.3 with HTTP/2", RFC 8740, DOI 10.17487/RFC8740, February 2020, <https://www.rfc-editor.org/info/rfc8740>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[RFC9155] Velvindron, L., Moriarty, K., and A. Ghedini, "Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2", RFC 9155, DOI 10.17487/RFC9155, December 2021, <https://www.rfc-editor.org/info/rfc9155>.

12.2. Informative References

[RFC5734] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Transport over TCP", STD 69, RFC 5734, DOI 10.17487/RFC5734, August 2009, <https://www.rfc-editor.org/info/rfc5734>.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC9205] Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <https://www.rfc-editor.org/info/rfc9205>.

Appendix A. Notes on Load Balancing

An EPP server should be able to serve a large number of concurrent requests from clients and return the responses in a fast and reliable manner. In addition, since EPP is extensible, EPP servers might be updated and the replacement of an EPP server with a new version should take place per the service level agreement negotiated between the registry and the registrars. To cost-effectively scale high volumes of requests and redeploy a server without affecting its functioning, best practice in providing a software service generally requires using load balancing. This section presents two possible approaches to the implementation of a HTTP load balancing solution for an EPP server.

An EPP server made up of a server pool must always operate with respect to the constraint that, once an EPP session is established, all the requests related to that session should be processed by the servers in the pool as long as the session is alive.

One possible approach is using sticky sessions. In this case, the load balancer assigns an identifier to each client issuing a request. Then, according to such identifier, the load balancer can route all of the requests of a given client to the backend server that started the session for its entire duration. This approach requires each backend server to maintain the EPP information connected to the sessions opened by that server. This means that when a backend server is stopped and then restarted after its update, all the sessions currently active and managed by that server are lost.

A more efficient solution consists in releasing the sessions from the server pool. According to this approach, every session is stored somewhere outside the server pool. The load balancer distributes the request based on the load of each backend server and according to a specific algorithm. When a server receives a request, it first retrieves the session information by the session ID and, if any, processes the request. Sessions are normally stored in a cluster of NO-SQL databases so that performance and efficiency requirements are fulfilled. In this approach, only the ongoing requests are lost when a backend server is stopped and restarted. Moreover, maintaining the sessions on a persistent data storage results in supporting a virtually unlimited number of concurrent sessions.

Authors' Addresses

Mario Loffredo
IIT-CNR/Registro.it
Via Moruzzi,1
56124 Pisa

Italy

Email: mario.loffredo@iit.cnr.it
URI: https://www.iit.cnr.it

Lorenzo Luconi Trombacchi
IIT-CNR/Registro.it
Via Moruzzi,1
56124 Pisa
Italy

Email: lorenzo.luconi@iit.cnr.it
URI: https://www.iit.cnr.it

Maurizio Martinelli
IIT-CNR/Registro.it
Via Moruzzi,1
56124 Pisa
Italy

Email: maurizio.martinelli@iit.cnr.it
URI: https://www.iit.cnr.it

Jan Romanowski
NASK/.pl Registry
Kolska 12
01-045 Warszawa
Poland

Email: jan.romanowski@nask.pl
URI: https://www.dns.pl/

Marcin Machnio
NASK/.pl Registry
Kolska 12
01-045 Warszawa
Poland

Email: info@dns.pl
URI: https://www.dns.pl/