draft-loffredo-regext-epp-over-http
M. Loffredo, L. Luconi Trombacchi, M. Martinelli
IIT-CNR/Registro.IT
J. Romanowski, M. Machnio
NASK/.pl Registry
Motivations of EPP over HTTP

• HTTP is loosely coupled with the network
• HTTP provides client-server cross-platform technology communication
• HTTP simplicity reduces the development time
• The speed gap between HTTP and TCP is actually not so large as in the past
• Load balancing can be more easily implemented at Layer 7 than at Layer 4
• Migrating an HTTP server to cloud requires less effort than a TCP server
Message Exchange

• A client **MUST** use the **HTTP POST** method to issue an EPP command through the request body

• A server receiving a request **MUST** return an EPP message in the response body

• The **Content-Length** header indicates the byte length of the entity body

• No EPP message information **MUST** be issued through any other part of the request or the response
The EPP session is implemented by using the mechanism described in RFC 6265.

A server receiving an EPP `<login>` command **MUST** use the "Set-Cookie" response header to send a token, a.k.a. session ID, that the client will return in future requests within the scope of the EPP session.

The name of the cookie attribute identifying the session ID is not relevant and depends on the implementations.

== Server -> Client ==
Set-Cookie: SID=52ceb07c2a824f09a1c6f9c45574097d

== Client -> Server ==
Cookie: SID=52ceb07c2a824f09a1c6f9c45574097d
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 response.

EPP sessions that are inactive for more than a server-defined period **MAY** be ended by the server.
<hello> Command

• A client **MAY** issue the `<hello>` command outside an EPP session

• In such case, the server **MUST** return the `<greeting>` response without starting a session:
  
  • no cookie is returned
  
  • an expired cookie is returned

• A client **MAY** issue the `<hello>` command within an EPP session (e.g. to keep it alive)
Return Codes

• HTTP error codes **MUST** be used for signaling HTTP requests failure

• EPP error codes **MUST** be used for signaling EPP commands failure

• The HTTP return code **200** is used for both successful and unsuccessful EPP requests
Implementations

• IIT-CNR/Registro.it EPP server

• NASK/.pl Registry EPP server
Security Considerations (1)

• HTTP over TLS (RFC 8740) **MUST** be used to protect sensitive information (e.g. credentials, authInfos, contact details) from disclosure while in transit.

• Servers are **RECOMMENDED** to implement additional measures to verify the client:
  
  • IP whitelisting
  
  • locking the session ID to the client's IP address

• Servers **MAY** require clients to present a valid X.509 digital certificate, issued by a recognized Certification Authority (CA), as described in RFC 8446.
Security Considerations (2)

• Session IDs **SHOULD** be randomly generated and be long enough to prevent them from being hijacked

• Servers **MAY**:
  
  • limit the lifetime of active sessions
  
  • control cookies usage by setting the cookie attributes (e.g. “Path”, “Max-Age”)

• Servers are **RECOMMENDED** to control the rate of both open EPP sessions and HTTP connections to mitigate the risk of resource starvation
Load Balancing (1)

• Using sticky sessions:

  • the load balancer assigns an identifier to each client issuing a request

  • 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

• Each backend server must maintain the EPP information about the sessions opened by that server

• When a backend server is stopped and then restarted, all the EPP sessions currently active are lost
Load Balancing (2)

- Releasing the sessions from the server pool:
  - every session is stored somewhere outside the server pool
  - the load balancer distributes the request based on the load of each backend server
  - when a server receives a request, it first retrieves the session data by the session ID
- Sessions are normally stored in a cluster of NO-SQL databases
- Only the ongoing requests are lost when a backend server is stopped and restarted
- Maintaining the sessions on a persistent data storage results in supporting a virtually unlimited number of concurrent sessions
Thanks for the attention!

Q & A