Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

M. Stubbig
Independent

October 30, 2017

Looking Glass API
draft-mst-lgapi-07

Abstract

This document introduces an application programming interface (API) to the web-based "Network Looking Glass" software. Its purpose is to provide application programmers uniform access to the Looking Glass service and to analyze standardized response.

The API interface is supposed to provide the same level of information as web-based interfaces, but in a computer-readable format.

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 May 3, 2018.

Table of Contents

1. Introduction

Many Internet service providers (ISPs) and Internet exchange points (IXPs) offer a complimentary web page to the general public, which gives insights to the backbone routing table, BGP neighbor information, or offered routes.

The visitor may also execute a ping or traceroute command to a random target address. Some ISPs even offer information about their multicast routing table including the mtrace command. The amount and type of offered information is not limited.

The service is mostly used for the purpose of troubleshooting and is known under the term of "Looking Glass". The development of various Looking Glass software has led to different systems in usage, syntax, style, and in offered information. The difference is of minor

interest to human users, but accessing a Looking Glass web page by script is complicated, inefficient, and error-prone.

Accessing a Looking Glass service by script is required for repeating tasks. It could be a monitoring system which validates link availability or bandwidth usage.

This leads to the requirement of a unified access method to the information provided by a Looking Glass. This document is a proposal of an application programmers interface (API), which provides a common schema, list of arguments, and options when accessing a Looking Glass service.

The transport protocol of choice is encrypted HTTP, the style of operation is REST, and the response format is JSON [RFC7159].

The Looking Glass API is described as a language-independent concept. Consequently, any programming language, which satisfies API commands listed in the following chapters, is acceptable.

1.1. Background

The requirement of a uniform access to a Looking Glass service becomes important when multiple Looking Glasses are part of a monitoring system. Implementing a web client and HTTP-parser for every kind of web-based Looking Glass is a time consuming workaround, however, the Looking Glass API is a much more viable, compatible, and scalable solution.

1.2. Terminology

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].

1.3. Syntax Notation

This specification uses the JavaScript Object Notation (JSON) of [RFC7159] arranged as JSend [JSend] compliant response.

1.4. Examples

All URLs in this documentation use the reserved sample domain of "example.net" as defined in [RFC6761] section 6.5.

IPv4 addresses use the documenational block of 192.0.2.0/24 [RFC5737] and IPv6 addresses reside in the reserved prefix of 2001:DB8::/32

[RFC3849]. BGP Autonomous System numbers are chosen from the private AS range defined in [RFC6996].

The printed examples skip some required parameters for reasons of simplicity.

2. Operation

An API client issues a query using the HTTP GET method to request a specific resources from the API server. The resource is a URI and can be informational or a command execution. The client must present all necessary parameters for the API server to execute the command on the selected router. Every API call is stateless and independent of the previous one.

The "API call" is a request from the client, which specifies a pre-defined operation ("API function") that the server will execute on a selected router. The "command" is a task executed on the router and initiated by the server on behalf of the client. The type and scope of all commands is defined and limited by the server. The client MUST NOT be able to execute random commands on the targeting router. There MUST NOT be any direct communication between the client and the router.

After the execution of the command on the selected router has finished, the server replies to the client if the operation has either succeeded, failed or timed out. The response is sent to the API client in JSON format. The communication protocol used between the server and router is not specified by this document; any method (e.g. Telnet, SSH, NETCONF, YANG, serial console) is acceptable.

All parameters and its values are case insensitive.

2.1. Method Parameters

Method parameters are mandatory components of the URI and placed in the "path" section in terms of [RFC7320]. Basically, the method parameters specify the API call and determine which command the client wants executed on the selected router.

2.2. Query Parameters

Query parameters are optional components of the URI and placed in the "query" section in terms of [RFC7320]. Generally, the query parameters are additional instructions for the requested command.

protocol

Restrict the command and method parameters to use the specified protocol and version. Protocol is selected as "Address Family Identifier" [IANA-AFN] [RFC2858] and optional "Subsequent Address Family Identifier" [IANA-SAFI] separated by comma.
Default value is 1,1 (IP version 4, unicast).
JSON data type is String.
Examples:

* protocol=2,1 (IP version 6, unicast)

* protocol=26 (MPLS, no SAFI used)

router
Run the command on the router identified by its name. This is not necessarily the routers hostname as long as the Looking Glass software recognizes it.
Default value is the first router in the list of available routers.
JSON data type is String.
Example: router=rbgn06.example.net

routerid
Run the command on this router identified by its position in the list of available routers.
Default value is "0".
JSON data type is Number.
Example: routerid=8

random
Append a random string to prevent the client (or an intermediate proxy) from caching the response. The API server must ignore its value.
No default value.
JSON data type is String.
Example: random=517A93B50

runtime
Stop executing the command after the runtime limit (in seconds) is exceeded. A value of 0 disables the limit.
Default value is "30".
JSON data type is Number.
Example: runtime=60

format
Request the server to provide the output (if any) in the selected format. Specify multiple formats separated by comma in descending order of preference. See Section 3.3.2 for more details.
Default value is "text/plain" (raw/unformatted output).

JSON data type is String.
Example: format=application/yang,text/plain

2.3. Response

The HTTP response header SHOULD set an appropriate HTTP status code as defined in [RFC7231] and MUST set the Content-Type to "application/json".

The HTTP body contains details and error descriptions. The response text MUST comply with the JSON syntax specification JSend [JSend], which is briefly explained in appendix JSend. Consequently every response MUST contain a "status" field of either "success", "fail", or "error", that are explained in the following sections.

2.3.1. Success

A successful response MUST set the "status" field to "success". It MUST also contain a "data" object including the following information:

performed_at
: combined date and time in UTC ISO 8601 [iso8601] indicating when the operation finished. This information MUST be present.

runtime
: amount of seconds (wallclock) used to run the command. This information MUST be present.

router
: the name of the router, that executed the command. This information MAY be present.

output
: output of the command in a format that was requested by the client or defaults to raw output as it appeared on the routers command line interface (CLI). It might even be blank if the command did not produce any output. This information SHOULD be present.

format
: selected output format by the server. The client might request multiple formats, so that the "Looking Glass" server has to choose the best option and tell the client which format was selected. This information SHOULD be present (or defaults to "text/plain" if missing).

Adding more information to the response is permitted and MUST be placed inside the "data" object.

The HTTP status code SHOULD be 200.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "status" : "success",
  "data" : {
    "router" : "route-server.lookingglass.example.net"
    "performed_at" : "2014-10-15T17:15:34Z",
    "runtime" : 2.63,
    "output" : [
      "full raw output from the observing router..."
    ],
    "format" : "text/plain"
  }
}
```

2.3.2. Fail

A status of "fail" indicates that the selected command was executed on the router but failed to succeed. The response message MUST set the "status" field to "fail" and MUST contain the "data" object with command-specific content, that is listed in Section 2.3.1.

The HTTP status code SHOULD be 200.

Example:

```
HTTP/2.0 200 OK
{
  "status" : "fail",
  "data" : {
    "performed_at" : "2014-10-18T20:04:37Z",
    "runtime" : 10.37,
    "output" : [
      "Sending 5, 100-byte ICMP Echos to 192.0.2.5",
      ".....",
      "Success rate is 0 percent (0/5)"
    ],
    "format" : "text/plain",
    "router" : "route-server.lookingglass.example.net"
  }
}
```

2.3.3. Error

The status "error" represents an error which occurred in processing the request or the command timed out. The response message MUST set the "status" field to "error" and MUST contain the "message" key, that keeps a meaningful message, explaining what went wrong.

The response MAY contain the "data" key, with required values listed in Section 2.3.1. It MAY also include a "code" field, that carries a numeric code corresponding to the error.

The HTTP status code SHOULD be 400 in case of a client-side error, 500 in case of a server-side error or 502 for errors occurring on the target router. Code 504 SHOULD be used when a command timed out.

Example:

```
HTTP/1.1 400 Bad Request
{
  "status" : "error",
  "message" : "Unrecognized host or address."
}
```

3. API functions

The Looking Glass API provides functions that are either mandatory or optional to implement. The same principle applies to the web-based Looking Glass.

It is not possible for any API function to modify the server's state. Therefore, all HTTP methods are GET operations.

Variables are templated and expanded in harmony of [RFC6570].

3.1. Diagnostic commands

3.1.1. Ping

Send echo messages to validate the reachability of a remote host and measure round-trip time. The "ping" command MAY use the ICMP protocol, but any protocol that satisfies the requirement is acceptable.

Implementation of the ping command is mandatory.

Syntax: https://lg.example.net/api/v1/ping/{addr}

Example:

https://lg.example.net/api/v1/ping/2001:DB8::35?protocol=2,1

3.1.2. Traceroute

Trace path from the executing router to the destination host and list all intermediate hops.

Implementation of the traceroute command is optional.

Syntax: https://lg.example.net/api/v1/traceroute/{addr}

Example:

https://lg.example.net/api/v1/traceroute/192.0.2.1?routerid=5

3.2. Informational commands

3.2.1. show route

Retrieve information about a specific subnet from the routing table.

Implementation of the "show route" command is mandatory.

Syntax: https://lg.example.net/api/v1/show/route/{addr}

Example:

https://lg.example.net/api/v1/show/route/2001:DB8::/48?protocol=2,1

3.2.2. show bgp

Display matching record from BGP routing table. This SHOULD include networks, next hop and MAY include metric, local preference, path list, weight, etc.

Implementation of the "show bgp" command is optional.

Syntax: https://lg.example.net/api/v1/show/bgp/{addr}

Example:

https://lg.example.net/api/v1/show/bgp/192.0.2.0/24

3.2.3. show bgp summary

Print a summary of BGP neighbor status. This MAY include neighbor BGP ID, autonomous system number, duration of peering, number of received prefixes, etc.

Implementation of the "show bgp summary" command is optional.

Syntax: https://lg.example.net/api/v1/show/bgp/summary

Example:

https://example.net/api/v1/show/bgp/summary?protocol=2&routerid=3

3.2.4. show bgp neighbors

Provide detailed information on BGP neighbor connections. Available details MAY include neighbor BGP ID, advertised networks, learned networks, autonomous system number, capabilities, protocol, statistics, etc.

Implementation of the "show bgp neighbors" command is optional.

Syntax: https://lg.example.net/api/v1/show/bgp/neighbors/{addr}

Example:

https://lg.example.net/api/v1/show/bgp/neighbors/192.0.2.226

3.3. Administrative commands

The following administrative commands MUST be included in the implementation.

3.3.1. router list

The command provides a full list of routers that are available for command execution. This list includes the router ID and its name. It is equivalent to the common "router" HTML drop-down form element and contains the same information.

Syntax: https://lg.example.net/api/v1/routers

Example response:

```
{
  "status" : "success",
  "data" : {
    "routers" : [
      "route-server.lookingglass.example.net",
      "customer-edge.lookingglass.example.net",
      "provider-edge.lookingglass.example.net"
    ],
    "performed_at" : "2014-10-19T20:07:01Z",
    "runtime" : 0.73
  }
}
```

3.3.2. router details

List additional information about the selected router, specified by its router ID. The response MUST contain the routers hostname and router ID. The response MAY contain more details like output format, country code, city, administrative contact, vendor and model.

Available output formats are specified by Internet media type as of [RFC6838] and listed in [IANA-MT]. If the routers supports multiple formats, they are separated by comma.

The router might provide output formats, that are not yet registered or listed in [IANA-MT]. RFC6838 [RFC6838] provides a tree for unregistered subtypes. For example, output in NETCONF format could use "text/x.netconf".

Missing output format defaults to "text/plain", which is a copy of the raw command-line output.

Syntax: https://lg.example.net/api/v1/routers/{number}

Example query:

https://lg.example.net/api/v1/routers/1

Example response:

```
{
  "status" : "success",
  "data" : {
    "id" : 1,
    "name" : "customer-edge.lookingglass.example.net",
    "format" : "text/plain,text/x.netconf",
    "country" : "de",
    "autonomous_system" : 64512
  }
}
```

3.3.3. commands

This function provides a full list of commands that are available for execution. The list includes mandatory, optional, and additional (Section 3.4) commands. It is equivalent to the "command" HTML drop-down or radio-button form element and contains the same information.

The list is formatted as "commands" array containing one object per command. This object contains informative strings about the current command: href, arguments, description and command.

Syntax: https://lg.example.net/api/v1/commands

Example response:

```
{
  "status" : "success",
  "data" : {
    "commands" : [
      {
        "href" : "https://lg.example.net/api/v1/show/route",
        "arguments" : "{addr}",
        "description" : "Print records from IP routing table",
        "command" : "show route"
      },
      {
        "href" : "https://lg.example.net/api/v1/traceroute",
        "arguments" : "{addr}",
        "description" : "Trace route to destination host",
        "command" : "traceroute"
      }
    ]
  }
}
```

3.4. Extensible commands

The list of commands MAY be expanded as long as the principles of this document are observed.

For example, a Looking Glass provider may not be offering BGP-related commands because of an OSPF-based network.

The sample command might be:

https://lg.example.net/api/v1/show/ospf/database

4. Miscellaneous

The network traffic sent by a "Looking Glass" is not appropriate when measuring Service Level Agreements or validating Quality of Service setting.

If a monitoring system uses the Looking Glass API for reachability checks, it SHOULD NOT rely on the HTTP status codes but on the "status" message field inside the HTTP body.

5. IANA Considerations

none

6. Security Consideration

The use of HTTPS is REQUIRED to ensure a high level of security, privacy, and confidentiality during transit.

6.1. Abuse Potential

The main goal of the Looking Glass API is the automated usage of the Looking Glass service. This allows the scripting of API calls, which could be used as a distributed Denial of Service (DDoS) attack. Interestingly, the attacked system recognizes the attack originating from various ISPs core network.

It is RECOMMENDED that implementers of the Looking Glass API take steps to mitigate the above described abuse. The strategy can include blocking or rate-limiting by client IP address or target IP network.

6.2. Authentication

Authentication is not a requirement because the current Looking Glass web services are usable without authentication. Requests to the proposed API service MAY be authenticated by any method. The decision is up to the implementers security requirements.

6.3. Minimal information

Some of the described commands provide a detailed insight into the providers network. It is therefore up to the implementer's security policy to dismiss commands that are marked as "optional" or restrict commands that are marked as "mandatory".

7. References

7.1. Normative References

[IANA-AFN] IANA, "Address Family Numbers", 2015, <https://www.iana.org/assignments/address-family-numbers/>.

[IANA-MT] IANA, "Media Types", 2015, <http://www.iana.org/assignments/media-types/media-types.xhtml>.

[IANA-SAFI] IANA, "Subsequent Address Family Identifier (SAFI) Parameters", 2015, <http://www.iana.org/assignments/safi-namespace/>.

[JSend] OmniTI Labs, "JSend", 2013, <http://labs.omniti.com/labs/jsend>.

[RFC2858] Bates, T., Rekhter, Y., Chandra, R., and D. Katz, "Multiprotocol Extensions for BGP-4", RFC 2858, DOI 10.17487/RFC2858, June 2000, <https://www.rfc-editor.org/info/rfc2858>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[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>.

7.2. Informative References

[iso8601] International Organization for Standardization, "Date and time format--ISO 8601", 2006, <http://www.iso.org/iso/iso8601>.

[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>.

[RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/info/rfc3849>.

[RFC5737] Arkko, J., Cotton, M., and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation", RFC 5737, DOI 10.17487/RFC5737, January 2010, <https://www.rfc-editor.org/info/rfc5737>.

[RFC6761] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", RFC 6761, DOI 10.17487/RFC6761, February 2013, <https://www.rfc-editor.org/info/rfc6761>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6996] Mitchell, J., "Autonomous System (AS) Reservation for Private Use", BCP 6, RFC 6996, DOI 10.17487/RFC6996, July 2013, <https://www.rfc-editor.org/info/rfc6996>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

Appendix A. JSend

According to [JSend], "JSend is a specification that lays down some rules for how JSON responses from web servers should be formatted. JSend focuses on application-level (as opposed to protocol- or

transport-level) messaging which makes it ideal for use in REST-style applications and APIs."

A basic JSend-compliant response MUST contain a "status" key and SHOULD contain "data", "message" and "code" keys dependent on the status value. The following table lists the required and optional keys.

Type	Required keys	Optional keys
success	status, data	
fail	status, data	
error	status, message	code, data

Table 1: Type and keys in JSend response

Author's Address

Markus Stubbig
Independent
Germany

Email: stubbig.ietf@gmail.com