pybatfish Documentation

Release 0.36.0

The Batfish Open Source Project

Mar 20, 2019

Contents

pybatfish is a library of Python bindings for Batfish. Using Python to analyze configurations requires access to a running Batfish service (which could be localhost). See here for building and running Batfish service.

CHAPTER 1

Getting started

To get started with Pybatfish, you will need a network snapshot. An example snapshot is packaged with Pybatfish (link) and can be used to step through the example below. Alternatively, you can package a snapshot of your own network as described here.

The following instructions show how to upload and query a network snapshot using Pybatfish in an interactive python shell like IPython. In these instructions, we assumed that Batfish is running on the same machine as Pybatfish, and the example snapshot included with Pybatfish is being analyzed.

1. Import Pybatfish:

```
>>> from pybatfish.client.commands import *
>>> from pybatfish.question.question import load_questions, list_questions
>>> from pybatfish.question import bfq
```

2. Load the question templates from the Batfish service into Pybatfish:

```
>>> load_questions()
```

4. Upload a network snapshot (you'll see some log messages followed by the name of initialized snapshot (prefixed by `ss_`):

```
>>> bf_init_snapshot('jupyter_notebooks/networks/example') # doctest: +ELLIPSIS
'ss_...'
```

Here, the example network is being uploaded, but this location could also be a folder or a zip containing a custom network snapshot.

5. Ask a question about the snapshot, using one of the loaded templates (`bfq` holds the questions currently loaded in Pybatfish). For example here, the question `IPOwners` fetches the mapping between IP address, interface, node and VRF for all devices in the network. :

```
>>> ip_owners_ans = bfq.ipOwners().answer()
```

`answer()` runs the question and returns the answer in a JSON format. See the Batfish questions directory for the set of questions that can be asked and their parameters.

6. To print the answer in a nice table, call `frame()` which wraps the answer as pandas dataframe. Calling head() on the dataframe will print the first 5 rows:

```
>>> ip_owners_ans.frame().head()
         Node      VRF           Interface           IP Mask Active
0    as2dist2  default           Loopback0     2.1.3.2   32   True
1    as2dist1  default           Loopback0     2.1.3.1   32   True
2    as2dept1  default  GigabitEthernet1/0  2.34.201.4   24   True
3    as2dept1  default           Loopback0     2.1.1.2   32   True
4  as3border2  default  GigabitEthernet1/0     3.0.2.1   24   True
```

7. Next, let's ask a question about interfaces. For example, to see all prefixes present on the interface `GigabitEthernet0/0` of the node `as1border1` we can use the `interfaceProperties` question like below:

```
>>> iface_ans = bfq.interfaceProperties(nodes='as1border1', interfaces=
↪'GigabitEthernet0/0', properties='all-prefixes').answer()
>>> iface_ans
                          Interface
    0  as1border1:GigabitEthernet0/0
```

For additional and more in-depth examples, check out the Jupyter Notebooks.

CHAPTER 2

Available questions

If you setup your `Batfish` service correctly, the questions outlined below should be available to you.

2.1 A note on types

1. You will see types such as `nodeSpec` and `interfacesSpec`. From a Python perspective, they are `string`. But they have a rich grammar underneath that enables flexible specification of a multiple nodes, interfaces etc. The grammar is outlined here grammar.

2. `headerConstraint` is a special type that allows you to place constraints on IPv4 packet header for questions that require it (e.g., *`traceroute`*) The *`HeaderConstraints`* class will help you quickly create such constraints.

2.2 List of questions

class pybatfish.question.bfq.**aaaAuthenticationLogin**(*, *nodes*, *question_name*)

Return nodes that do not require authentication on all lines.

Parameters **nodes** (*nodeSpec*) – *Required.* Examine AAA Authentication on nodes matching this name or regex.

Default value: `.*`

class pybatfish.question.bfq.**bgpPeerConfiguration**(*, *nodes*, *properties*, *question_name*)

Return BGP peer configuration properties.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*bgpPeerPropertySpec*) – Include properties matching this regex.

class pybatfish.question.bfq.**bgpProcessConfiguration**(*, *nodes*, *properties*, *question_name*)

Return BGP process configuration properties.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*bgpProcessPropertySpec*) – Include properties matching this regex.

class pybatfish.question.bfq.**bgpSessionCompatibility**(*, *nodes*, *remoteNodes*, *status*, *type*, *question_name*)

Return the status of configured BGP sessions, independent of remote peer configurations.

Parameters

- **nodes** (*nodeSpec*) – Include sessions whose first node matches this specifier.
- **remoteNodes** (*nodeSpec*) – Include sessions whose second node matches this specifier.
- **status** (bgpSessionStatus) – Only include sessions for which status matches this regex.
- **type** (*bgpSessionType*) – Only include sessions for which type (ibgp, ebgp_singlehop, ebgp_multihop) matches this regex.

class pybatfish.question.bfq.**bgpSessionStatus**(*, *nodes*, *remoteNodes*, *status*, *type*, *question_name*)

Return the status of configured BGP sessions.

Parameters

- **nodes** (*nodeSpec*) – Include sessions whose first node matches this specifier.
- **remoteNodes** (*nodeSpec*) – Include sessions whose second node matches this specifier.
- **status** (bgpSessionStatus) – Only include sessions for which status matches this regex.
- **type** (*bgpSessionType*) – Only include sessions for which type (ibgp, ebgp_singlehop, ebgp_multihop) matches this regex.

class pybatfish.question.bfq.**bidirectionalReachability**(*, *headers*, *pathConstraints*, *returnFlowType*, *question_name*)

Search for successfully delivered flows that can successfully receive a response.

Performs two reachability analyses, first originating from specified sources, then returning back to those sources. After the first (forward) pass, sets up sessions in the network and creates returning flows for each successfully delivered forward flow. The second pass searches for return flows that can be successfully delivered in the presence of the setup sessions.

Parameters

- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).
- **returnFlowType** (*string*) – Specifies whether to search for return flows that are successful (SUCCESS), that fail (FAILURE), or that can either succeed or fail depending on the path taken (MULTIPATH_INCONSISTENT). The default is SUCCESS.

class pybatfish.question.bfq.**bidirectionalTraceroute**(*, *headers*, *startLocation*, *ignoreFilters*, *maxTraces*, *question_name*)

Trace the path(s) for the specified flow, along with path(s) for reverse flows.

This question performs a virtual traceroute in the network from a starting node. A destination IP and ingress (source) node must be specified. Other IP headers are given default values if unspecified. If the trace succeeds, a traceroute is performed in the reverse direction.

Parameters

- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **startLocation** (*locationSpec*) – *Required.* Location (node and interface combination) to start tracing from.
- **ignoreFilters** (*boolean*) – If set, filters/ACLs encountered along the path are ignored.
- **maxTraces** (*integer*) – Limit the number of traces returned.

class pybatfish.question.bfq.**compareFilters**(*, *nodes*, *filters*, *question_name*)

Compares filters with the same name in the current and reference snapshots. Returns pairs of lines, one from each filter, that match the same flow(s) but treat them differently (i.e. one permits and the other denies the flow).

This question can be used to summarize how a filter has changed over time. In particular, it highlights differences that cause flows to be denied when they used to be permitted, or vice versa. The output is a table that includes pairs of lines, one from each version of the filter, that both match at least one common flow, and have different action (permit or deny).

Parameters

- **nodes** (*nodeSpec*) – Only evaluate filters present on nodes matching this specifier.
- **filters** (*filter*) – Only evaluate filters that match this regex.

class pybatfish.question.bfq.**definedStructures**(*, *names*, *nodes*, *types*, *question_name*)

Lists the structures defined in the network.

Lists the structures defined in the network, along with the files and line numbers in which they are defined.

Parameters

- **names** (*structureName*) – Include structures whose name matches this string or regex.

  Default value: `.*`
- **nodes** (*nodeSpec*) – Include files used to generate nodes whose name matches this specifier.

  Default value: `.*`
- **types** (*namedStructureSpec*) – Include structures whose vendor-specific type matches this specifier.

  Default value: `.*`

class pybatfish.question.bfq.**detectLoops**(*, *maxTraces*, *question_name*)

Detect forwarding loops.

Finds forwarding loops.

Parameters **maxTraces** (*integer*) – Limit the number of traces returned.

class pybatfish.question.bfq.**differentialReachability**(*, *actions*, *headers*, *ignoreFilters*, *invertSearch*, *maxTraces*, *pathConstraints*, *question_name*)

Detect differential reachability.

Finds flows that are accepted in one snapshot but dropped in another.

Parameters

- **actions** (*dispositionSpec*) – Only return flows for which the disposition is from this set.

  Default value: `success`
- **headers** (*headerConstraint*) – Packet header constraints.
- **ignoreFilters** (*boolean*) – Do not apply filters/ACLs during analysis.
- **invertSearch** (*boolean*) – Search for packet headers outside the specified headerspace, rather than inside the space.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).

class pybatfish.question.bfq.**edges**(*, *nodes*, *remoteNodes*, *edgeType*, *question_name*)

Lists different types of edges in a snapshot.

Lists neighbor relationships of the specified type (layer3, BGP, ospf, etc. in the form of edges).

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include edges whose first node matches this name or regex.

  Default value: `.*`
- **remoteNodes** (*nodeSpec*) – *Required.* Include edges whose second node matches this name or regex.

  Default value: `.*`
- **edgeType** (*string*) – Types of edges to include. Allowed values:
  - bgp
  - eigrp
  - ipsec
  - isis
  - layer1
  - layer2
  - layer3
  - ospf
  - rip
  - vxlan

class pybatfish.question.bfq.**f5BigipVipConfiguration**(*, *nodes*, *question_name*)

Report VIP mappings of F5 BIG-IP configurations.

Parameters **nodes** (*nodeSpec*) – Include nodes matching this name or regex.

class pybatfish.question.bfq.**fileParseStatus**(*, *question_name*)
Display file parse status.

For each file in a snapshot, returns the host(s) that were produced by the file and the parse status: pass, fail, partially parsed.

class pybatfish.question.bfq.**filterLineReachability**(*, *filters*, *ignoreComposites*, *nodes*, *question_name*)
Identify ACLs/filters with unreachable lines.

This question finds all unreachable lines in the specified ACLs/filters.

Parameters

- **filters** (*filterSpec*) – Specifier for filters to test.
- **ignoreComposites** (*boolean*) – Whether to ignore filters that are composed of multiple filters defined in the configs.
- **nodes** (*nodeSpec*) – Examine filters on nodes matching this specifier.

class pybatfish.question.bfq.**filterTable**(*, *innerQuestion*, *columns*, *filter*, *question_name*)
Return subset of answer for a question.

Return a subset of the answer generated by the inner question. The results are trimmed first by row and then by column. Rows where any value matches the filter are returned. The columns returned for each row is restricted by the column specifier.

Parameters

- **innerQuestion** (*question*) – *Required.* The inner question whose answer should be filtered.
- **columns** (*string*) – The set of columns to fetch.
- **filter** (*string*) – The filter to use.

class pybatfish.question.bfq.**initIssues**(*, *question_name*)
Return issues that occurred when parsing input files and converting to vendor independent model.

class pybatfish.question.bfq.**interfaceMtu**(*, *interfaces*, *mtuBytes*, *nodes*, *comparator*, *question_name*)
Find interfaces where the configured MTU matches the specified comparator and mtuBytes.

For example, if comparator is '<' and mtuBytes is 1500, then only interfaces where the configured MTU is less than 1500 bytes will be returned.

Parameters

- **interfaces** (*interfacesSpec*) – *Required.* Evaluate interfaces matching this specifier.

  Default value: `.*`
- **mtuBytes** (*integer*) – *Required.* The reference MTU in bytes against which to check the configured MTU.

  Default value: `1500`
- **nodes** (*nodeSpec*) – *Required.* Include nodes matching this specifier.

  Default value: `.*`

- **comparator** (*comparator*) – Returned devices will satisfy <comparator> <mtuBytes>. Use '<' to find devices that do not have MTU smaller than the specified <mtuBytes> MTU.

  Default value: <

class pybatfish.question.bfq.**interfaceProperties**(*, *excludeShutInterfaces*, *interfaces*, *nodes*, *properties*, *question_name*)

Returns configuration properties of interfaces.

Parameters

- **excludeShutInterfaces** (*boolean*) – Exclude interfaces that are shutdown.
- **interfaces** (*interfacesSpec*) – Include interfaces matching this specifier.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **properties** (*interfacePropertySpec*) – Include properties matching this specifier.

class pybatfish.question.bfq.**ipOwners**(*, *duplicatesOnly*, *question_name*)

Returns the mapping of IP address, interface, node and VRF for all devices in the snapshot.

Parameters **duplicatesOnly** (*boolean*) – *Required.* Restrict output to only IP addresses that are duplicated (configured on a different node or VRF) in the snapshot.

class pybatfish.question.bfq.**ipsecSessionStatus**(*, *nodes*, *remoteNodes*, *status*, *question_name*)

Returns the status of configured IPSec sessions.

Parameters

- **nodes** (*nodeSpec*) – Include sessions whose first node matches this specifier.
- **remoteNodes** (*nodeSpec*) – Include sessions whose second node matches this specifier.
- **status** (ipsecSessionStatus) – Only include IPSec sessions for which status matches this regex.

class pybatfish.question.bfq.**loopbackMultipathConsistency**(*, *maxTraces*, *question_name*)

Validate multipath consistency between all pairs of loopbacks.

Finds flows between loopbacks that are treated differently by different paths in the presence of multipath routing.

Parameters **maxTraces** (*integer*) – Limit the number of traces returned.

class pybatfish.question.bfq.**lpmRoutes**(*, *ip*, *nodes*, *vrfs*, *question_name*)

Show routes which are longest prefix match for a given IP address.

Return longest prefix match routes for a given IP in the RIBs of specified node/VRF.

Parameters

- **ip** (*ip*) – *Required.* IP address to run LPM on.
- **nodes** (*nodeSpec*) – *Required.* Examine routes on nodes matching this specifier.

  Default value: `.*`
- **vrfs** (*vrf*) – *Required.* Examine routes on VRFs matching this name or regex.

  Default value: `.*`

class pybatfish.question.bfq.**mlagProperties**(*, *idRegex*, *nodes*, *question_name*)

Returns configuration properties of MLAGs.

Parameters

- **idRegex** (*javaRegex*) – Include MLAG IDs matching this java Regex.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.

class pybatfish.question.bfq.**multipathConsistency**(*, *headers*, *maxTraces*, *pathConstraints*, *question_name*)

Validate multipath consistency.

Finds flows that are treated differently by different paths in the presence of multipath routing.

Parameters

- **headers** (*headerConstraint*) – Packet header constraints.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).

class pybatfish.question.bfq.**namedStructures**(*, *ignoreGenerated*, *indicatePresence*, *nodes*, *structureNames*, *structureTypes*, *question_name*)

Return named structure definitions.

Parameters

- **ignoreGenerated** (*boolean*) – Whether to ignore auto-generated structures.

  Default value: `True`
- **indicatePresence** (*boolean*) – Output if the structure is present or absent.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **structureNames** (*structureName*) – Include structures matching this name or regex.
- **structureTypes** (*namedStructureSpec*) – Include structures of this type.

class pybatfish.question.bfq.**neighbors**(*, *nodes*, *remoteNodes*, *style*, *neighborTypes*, *roleDimension*, *question_name*)

Lists neighbor relationships in the snapshot.

Lists neighbor relationships of the specified type (layer3, eBGP, iBGP, ospf, etc.).

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include neighbors whose first node matches this specifier.

  Default value: `.*`
- **remoteNodes** (*nodeSpec*) – *Required.* Include neighbors whose second node matches this specifier.

  Default value: `.*`
- **style** (*string*) – *Required.* String indicating the style of information requested about each neighbor. Allowed values:

  – role

  – summary

  – verbose

  Default value: `summary`

- **neighborTypes** (*string*) – Types of neighbor relationships to include. Allowed values:
  - ebgp
  - eigrp
  - ibgp
  - layer1
  - layer2
  - layer3
  - ospf
  - rip
- **roleDimension** (*nodeRoleDimension*) – Role dimension to run the question on.

class pybatfish.question.bfq.**nodeProperties**(*, *nodes*, *properties*, *question_name*)
Return configuration properties of nodes.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*nodePropertySpec*) – Include properties matching this regex.

class pybatfish.question.bfq.**nodes**(*, *nodes*, *summary*, *nodeTypes*, *question_name*)
Lists configuration attributes of nodes in the network.

Returns a JSON dictionary with all (or summary) of the configuration parameters that are stored in the vendor independent data-model.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include nodes matching this name or regex.

  Default value: `.*`
- **summary** (*boolean*) – *Required.* Whether to provide only summary information about each node rather than the full data model.

  Default value: `True`
- **nodeTypes** (*string*) – Include nodes of the specified types.

class pybatfish.question.bfq.**ospfAreaConfiguration**(*, *nodes*, *question_name*)
Return configuration parameters of OSPF areas.

Parameters **nodes** (*nodeSpec*) – Include nodes matching this name or regex.

class pybatfish.question.bfq.**ospfInterfaceConfiguration**(*, *nodes*, *properties*, *question_name*)
Returns OSPF configuration of interfaces.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*ospfPropertySpec*) – Include properties matching this regex.

class pybatfish.question.bfq.**ospfProcessConfiguration**(*, *nodes*, *properties*, *question_name*)
Return configuration parameters for OSPF routing processes.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*ospfPropertySpec*) – Include properties matching this regex.

class pybatfish.question.bfq.**ospfProperties**(*, *nodes*, *properties*, *question_name*)
Return configuration parameters for OSPF routing processes.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*ospfPropertySpec*) – Include properties matching this regex.

class pybatfish.question.bfq.**ospfSessionCompatibility**(*, *nodes*, *remoteNodes*, *question_name*)
Returns compatible OSPF sessions.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **remoteNodes** (*nodeSpec*) – Include remote nodes matching this name or regex.

class pybatfish.question.bfq.**parseWarning**(*, *aggregateDuplicates*, *question_name*)
Return a table of the Batfish warnings that occurred parsing this snapshot.

Parameters **aggregateDuplicates** (*boolean*) – Whether to aggregate duplicate results.

class pybatfish.question.bfq.**prefixTracer**(*, *nodes*, *prefix*, *question_name*)
Trace prefix propagation through the network.

Parameters

- **nodes** (*nodeSpec*) – Include prefix tracing information for nodes matching this name or regex.
- **prefix** (*prefix*) – The prefix to trace. Expected format is A.B.C.D/Y.

class pybatfish.question.bfq.**reachability**(*, *pathConstraints*, *headers*, *actions*, *maxTraces*, *invertSearch*, *ignoreFilters*, *question_name*)
Find flows that match the ingress and egress location, src and dest ip address and disposition constraints, as described by the input specifiers.

Parameters

- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).
- **headers** (*headerConstraint*) – Packet header constraints.
- **actions** (*dispositionSpec*) – Only return flows for which the disposition is from this set.

  Default value: `success`
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **invertSearch** (*boolean*) – Search for packet headers outside the specified headerspace, rather than inside the space.
- **ignoreFilters** (*boolean*) – Do not apply filters/ACLs during analysis.

class pybatfish.question.bfq.**referencedStructures**(*, *names*, *nodes*, *types*, *question_name*)
Lists the references in configuration files to vendor-specific structures.

Lists the references in configuration files to vendor-specific structures, along with the line number, the name and the type of the structure referenced, and configuration context in which each reference occurs.

Parameters

- **names** (*structureName*) – Include structures whose name matches this string or regex.

  Default value: `.*`

- **nodes** (*nodeSpec*) – Include files used to generate nodes whose name matches this specifier.

  Default value: `.*`

- **types** (*namedStructureSpec*) – Include structures whose vendor-specific type matches this specifier.

  Default value: `.*`

class pybatfish.question.bfq.**resolveFilterSpecifier**(*, *filters*, *grammarVersion*, *nodes*, *question_name*)

Show the resolved values for filter specifier.

Parameters

- **filters** (*filterSpec*) – Input to the FilterSpecifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **nodes** (*nodeSpec*) – Input to the NodeSpecifier.

class pybatfish.question.bfq.**resolveInterfaceSpecifier**(*, *grammarVersion*, *interfaces*, *nodes*, *question_name*)

Show the resolved values for interface specifier.

Parameters

- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **interfaces** (*interfacesSpec*) – Input to the interfaceSpecifier.
- **nodes** (*nodeSpec*) – Input to the NodeSpecifier.

class pybatfish.question.bfq.**resolveIpSpecifier**(*, *grammarVersion*, *ips*, *question_name*)

Show the resolved values for IP space specifier.

Parameters

- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **ips** (*ipSpaceSpec*) – Input to the IP space specifier.

class pybatfish.question.bfq.**resolveIpsOfLocationSpecifier**(*, *grammarVersion*, *locations*, *question_name*)

Show IPs that are auto-assigned to locations.

Parameters

- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **locations** (*locationSpec*) – Input to the LocationSpecifier.

class pybatfish.question.bfq.**resolveLocationSpecifier**(*, *grammarVersion*, *locations*, *question_name*)

Show the resolved values for location specifier.

Parameters

- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **locations** (*locationSpec*) – Input to the LocationSpecifier.

class pybatfish.question.bfq.**resolveNodeSpecifier**(*, *grammarVersion*, *nodes*, *question_name*)

Show the resolved values for node specifier.

Parameters

- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **nodes** (*nodeSpec*) – Input to the NodeSpecifier.

class pybatfish.question.bfq.**routes**(*, *nodes*, *vrfs*, *network*, *protocols*, *rib*, *question_name*)

Show routing tables.

Return routes for the specified RIB for specified VRF for specified node(s).

Parameters

- **nodes** (*nodeSpec*) – *Required.* Examine routes on nodes matching this specifier.

  Default value: `.*`
- **vrfs** (*vrf*) – *Required.* Examine routes on VRFs matching this name or regex.

  Default value: `.*`
- **network** (*prefix*) – Examine routes for networks matching this prefix.
- **protocols** (*routingProtocolSpec*) – Examine routes for protocols matching this specifier.
- **rib** (*string*) – Only return routes from a given protocol RIB. Allowed values:
  - main
  - bgp

class pybatfish.question.bfq.**searchFilters**(*, *action*, *explain*, *filters*, *headers*, *invertSearch*, *nodes*, *startLocation*, *question_name*)

Find flows for which a filter takes a particular behavior.

This question searches for flows for which a filter (access control list) has a particular behavior. The behaviors can be: that the filter permits the flow (permit), that it denies the flow (deny), or that the flow is matched by a particular line (matchLine <lineNumber>). Filters are selected using node and filter specifiers, which might match multiple filters. In this case, a (possibly different) flow will be found for each filter.

Parameters

- **action** (*string*) – The behavior that you want evaluated. Options are: permit|deny|matchLine <line number>. Only one option should be selected.
- **explain** (*boolean*) – Include a description of the flow space matching the query.
- **filters** (*filterSpec*) – Only evaluate filters that match this specifier.
- **headers** (*headerConstraint*) – Packet header constraints on the flows being searched.
- **invertSearch** (*boolean*) – Search for packet headers outside the specified headerspace, rather than inside the space.
- **nodes** (*nodeSpec*) – Only evaluate filters present on nodes matching this specifier.

- **startLocation** (*locationSpec*) – Only consider specified locations as possible sources.

class pybatfish.question.bfq.**subnetMultipathConsistency**(*, *maxTraces*, *question_name*)

Validate multipath consistency between all pairs of subnets.

Finds flows between subnets that are treated differently by different paths in the presence of multipath routing.

Parameters **maxTraces** (*integer*) – Limit the number of traces returned.

class pybatfish.question.bfq.**switchedVlanProperties**(*, *excludeShutInterfaces*, *interfaces*, *nodes*, *vlans*, *question_name*)

Returns configuration properties of switched VLANs.

Parameters

- **excludeShutInterfaces** (*boolean*) – Exclude interfaces that are shutdown.
- **interfaces** (*interfacesSpec*) – Include interfaces matching this specifier.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **vlans** (*integerSpace*) – Include VLANs in this space.

class pybatfish.question.bfq.**testFilters**(*, *filters*, *headers*, *nodes*, *startLocation*, *question_name*)

Evaluate the processing of a flow by a given filter/ACL.

Find how the specified flow is processed through the specified filters/ACLs.

Parameters

- **filters** (*filterSpec*) – *Required.* Only consider filters that match this specifier.

  Default value: `.*`
- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **nodes** (*nodeSpec*) – *Required.* Only examine filters on nodes matching this specifier.

  Default value: `.*`
- **startLocation** (*string*) – Location to start tracing from.

class pybatfish.question.bfq.**traceroute**(*, *startLocation*, *headers*, *maxTraces*, *ignoreFilters*, *question_name*)

Trace the path(s) for the specified flow.

This question performs a virtual traceroute in the network from a starting node. A destination IP and ingress (source) node must be specified. Other IP headers are given default values if unspecified. Unlike a real traceroute, this traceroute is directional. That is, for it to succeed, the reverse connectivity is not needed. This feature can help debug connectivity issues by decoupling the two directions.

Parameters

- **startLocation** (*locationSpec*) – *Required.* Location (node and interface combination) to start tracing from.
- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **ignoreFilters** (*boolean*) – If set, filters/ACLs encountered along the path are ignored.

class pybatfish.question.bfq.**undefinedReferences**(*, *nodes*, *question_name*)
Identify undefined references in configuration.

This question finds configurations that have references to named structures (e.g., ACLs) that are not defined. Such occurrences indicate errors and can have serious consequences in some cases.

Parameters **nodes** (*nodeSpec*) – *Required.* Look for undefined references on nodes matching this name or regex.

Default value: `.*`

class pybatfish.question.bfq.**unusedStructures**(*, *nodes*, *question_name*)
Return nodes with structures such as ACLs, routemaps, etc. that are defined but not used.

Return nodes with structures such as ACLs, routes, etc. that are defined but not used. This may represent a bug in the configuration, which may have occurred because a final step in a template or MOP was not completed. Or it could be harmless extra configuration generated from a master template that is not meant to be used on those nodes.

Parameters **nodes** (*nodeSpec*) – *Required.* Look for unused structures on nodes matching this name or regex.

Default value: `.*`

class pybatfish.question.bfq.**viConversionStatus**(*, *question_name*)
Display vendor independent conversion status.

For each node in a snapshot, returns the vendor independent conversion status: pass, fail, converted with warnings.

class pybatfish.question.bfq.**viConversionWarning**(*, *question_name*)
Return Batfish warnings that occurred when converting to vendor independent model.

class pybatfish.question.bfq.**viModel**(*, *question_name*)
Lists configuration attributes of nodes and edges in the network.

Returns a JSON dictionary with all of the configuration parameters and neighbor relations stored in the vendor independent data-model.

class pybatfish.question.bfq.**vxlanVniProperties**(*, *nodes*, *properties*, *question_name*)
Returns configuration properties of VXLANs.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **properties** (*vxlanVniPropertySpec*) – Include properties matching this specifier.

CHAPTER 3

Datamodel classes

Here we describe classes used in answers and their attributes, which may help you filter your answers as desired.

3.1 Base types

class pybatfish.datamodel.primitives.**Assertion**(*type: pybatfish.datamodel.primitives.AssertionType*, *expect*)

A Batfish assertion.

Assertions are combined with a `Question` to create a Batfish check. An assertion can be on the number of results return by the question, or on the value of the answer itself.

Variables

- **type** – an *AssertionType*
- **expect** – the expected value (a.k.a as right-hand side) for the assertion to return True.

class pybatfish.datamodel.primitives.**AssertionType**

Assertion type.

COUNT_EQUALS = 'countequals'

Number of results equals

COUNT_LESSTHAN = 'countlessthan'

Number of results is less than

COUNT_MORETHAN = 'countmorethan'

Number of results is more than

EQUALS = 'equals'

Result equals to value (list of rows). **Experimental**

class pybatfish.datamodel.primitives.**VariableType**

Auto completion type.

ADDRESS_GROUP_AND_BOOK = 'addressGroupAndBook'
address group and book pair

ANSWER_ELEMENT = 'answerElement'
answer elements

APPLICATION_SPEC = 'applicationSpec'
application specifier

BGP_PEER_PROPERTY_SPEC = 'bgpPeerPropertySpec'
bgp peer properties

BGP_PROCESS_PROPERTY_SPEC = 'bgpProcessPropertySpec'
bgp process properties

BGP_SESSION_STATUS = 'bgpSessionStatus'
bgp session statuses

BGP_SESSION_TYPE = 'bgpSessionType'
bgp session types

BOOLEAN = 'boolean'
boolean values

COMPARATOR = 'comparator'
comparators (<, <=, ==, >=, >, !=)

DISPOSITION_SPEC = 'dispositionSpec'
dispositions

DOUBLE = 'double'
double values

FILTER = 'filter'
names or regex of filters

FILTER_NAME = 'filter'
name of filters

FILTER_SPEC = 'filterSpec'
filter specifier

FLOAT = 'float'
float values

FLOW_STATE = 'flowState'
flow states

HEADER_CONSTRAINT = 'headerConstraint'
packet header constraints

INTEGER = 'integer'
integer values

INTEGER_SPACE = 'integerSpace'
integer spaces

INTERFACE = 'interface'
names of interfaces

INTERFACES_SPEC = 'interfacesSpec'
interfaces specifier

INTERFACE_GROUP_AND_BOOK = 'interfaceGroupAndBook'
interface group, book

INTERFACE_NAME = 'interfaceName'
name of interfaces

INTERFACE_PROPERTY_SPEC = 'interfacePropertySpec'
interface properties

IP = 'ip'
ips

IPSEC_SESSION_STATUS = 'ipsecSessionStatus'
ipsec session statuses

IP_PROTOCOL = 'ipProtocol'
ip protocols

IP_PROTOCOL_SPEC = 'ipProtocolSpec'
ip protocol specifier

IP_SPACE_SPEC = 'ipSpaceSpec'
ip space specifier

IP_WILDCARD = 'ipWildcard'
ip protocols

JAVA_REGEX = 'javaRegex'
java regex

JSON_PATH = 'jsonPath'
json path

JSON_PATH_REGEX = 'jsonPathRegex'
json path regex

LOCATION_SPEC = 'locationSpec'
location specifier

LONG = 'long'
long values

NAMED_STRUCTURE_SPEC = 'namedStructureSpec'
named structure type

NODE_NAME = 'nodeName'
name of nodes

NODE_PROPERTY_SPEC = 'nodePropertySpec'
node properties

NODE_ROLE_AND_DIMENSION = 'nodeRoleAndDimension'
node role,dimension

NODE_ROLE_DIMENSION = 'nodeRoleDimension'
names of node role dimensions

NODE_SPEC = 'nodeSpec'
node specifier

OSPF_PROPERTY_SPEC = 'ospfPropertySpec'
ospf properties

PATH_CONSTRAINT = 'pathConstraint'
path constraints

PREFIX = 'prefix'
prefixes

PREFIX_RANGE = 'prefixRange'
prefix ranges

PROTOCOL = 'protocol'
application-level protocols

QUESTION = 'question'
questions

ROUTING_PROTOCOL_SPEC = 'routingProtocolSpec'
routing protocols

STRING = 'string'
string values

STRUCTURE_NAME = 'structureName'
names of structures

SUBRANGE = 'subrange'
subranges

VRF = 'vrf'
names of vrfs

VXLAN_VNI_PROPERTY_SPEC = 'vxlanVniPropertySpec'
vxlan vni properties

ZONE = 'zone'
names of zones

class pybatfish.datamodel.primitives.**AutoCompleteSuggestion**(*description: str*, *insertion_index: int*, *is_partial: bool*, *rank: int*, *text: str*)

Represent one auto complete suggestion.

Auto complete suggestions are returned by Batfish for auto complete queries.

Variables

- **description** – A description of the suggestion (optional)
- **insertion_index** – Index in original input string where suggested text should be inserted
- **is_partial** – Whether this suggestion represents partial or full text
- **rank** – Batfish may assign a rank to the suggestion
- **text** – The actual suggested text

class pybatfish.datamodel.primitives.**Edge**(*node1: str*, *node1interface*, *node2: str*, *node2interface*)

A network edge (i.e., a link between two node/interface pairs).

Variables

- **node1** – First node name
- **node1interface** – First node's interface name

- **node2** – Second node name
- **node2interface** – Second node's interface name

class pybatfish.datamodel.primitives.**FileLines**(*filename: str*, *lines: List[int] = NOTHING*)

A class that represents a set of lines in a file.

Variables

- **filename** – The filename referenced
- **lines** – A list of lines referenced

class pybatfish.datamodel.primitives.**Interface**(*hostname: str*, *interface: str*)

A network interface — a combination of node and interface names.

Variables

- **hostname** – Node hostname to which this interface belongs
- **interface** – Interface name

class pybatfish.datamodel.primitives.**Issue**(*severity*, *explanation: str = NOTHING*, *type: pybatfish.datamodel.primitives.IssueType = NOTHING*)

Information about a bug/issue that Batfish has discovered.

Variables

- **severity** – The integer severity of the issue
- **explanation** – An explanation for the issue
- **type** – An *IssueType* containing more information about the issue

class pybatfish.datamodel.primitives.**IssueType**(*major: str*, *minor: str*)

Details about a particular *Issue* type.

Variables

- **major** – Primary type of the issue
- **minor** – Additional subcategory of the issue

class pybatfish.datamodel.primitives.**ListWrapper**

Helper list class that implements _repr_html_().

3.2 ACL traces

class pybatfish.datamodel.acl.**AclTrace**(*events: List[pybatfish.datamodel.acl.AclTraceEvent] = NOTHING*)

The trace of a packet's life through an ACL.

Variables **events** – A list of *AclTraceEvent*

class pybatfish.datamodel.acl.**AclTraceEvent**(*class_name: Optional[str] = None*, *description: Optional[str] = None*, *lineDescription: Optional[str] = None*)

One event corresponding to a packet's life through an ACL.

Variables

- **class_name** – The type of the event that occurred while tracing.

- **description** – The description of the event
- **lineDescription** – ACL line that caused the event (if applicable)

3.3 Flows and packets

class pybatfish.datamodel.flow.**EnterInputIfaceStepDetail**(*inputInterface: str, inputVrf: Optional[str]*)

Details of a step representing the entering of a flow into a Hop.

Variables

- **inputInterface** – Interface of the Hop on which this flow enters
- **inputVrf** – VRF associated with the input interface

class pybatfish.datamodel.flow.**ExitOutputIfaceStepDetail**(*outputInterface: str, transformedFlow: Optional[str]*)

Details of a step representing the exiting of a flow out of a Hop.

Variables

- **outputInterface** – Interface of the Hop from which the flow exits
- **transformedFlow** – Transformed Flow if a source NAT was applied on the Flow

class pybatfish.datamodel.flow.**FilterStepDetail**(*filter: str*, *filterType: str*)

Details of a step representing a filter step.

Variables

- **filter** – filter name
- **type** – filter type

class pybatfish.datamodel.flow.**Flow**(*dscp, dstIp, dstPort, ecn, fragmentOffset, icmpCode, icmpVar, ingressInterface: Optional[str], ingressNode: Optional[str], ingressVrf: Optional[str], ipProtocol: str, packetLength: str, srcIp, srcPort, state, tag, tcpFlagsAck, tcpFlagsCwr, tcpFlagsEce, tcpFlagsFin, tcpFlagsPsh, tcpFlagsRst, tcpFlagsSyn, tcpFlagsUrg*)

A concrete IPv4 flow.

Noteworthy attributes for flow inspection/filtering:

Variables

- **srcIP** – Source IP of the flow
- **dstIP** – Destination IP of the flow
- **srcPort** – Source port of the flow
- **dstPort** – Destination port of the flow
- **ipProtocol** – the IP protocol of the flow (as integer, e.g., 1=ICMP, 6=TCP, 17=UDP)
- **ingressNode** – the node where the flow started (or entered the network)
- **ingressInterface** – the interface name where the flow started (or entered the network)
- **ingressVrf** – the VRF name where the flow started (or entered the network)

class pybatfish.datamodel.flow.**FlowTrace**(*disposition*, *hops*, *notes*)
A trace of a flow through the network.

A flowTrace is a combination of hops and flow fate (i.e., disposition).

Variables

- **disposition** – Flow disposition
- **hops** – A list of hops (*FlowTraceHop*) the flow took
- **notes** – Additional notes that help explain the disposition, if applicable.

class pybatfish.datamodel.flow.**FlowTraceHop**(*edge: pybatfish.datamodel.primitives.Edge, routes: List[Any], transformedFlow: Optional[pybatfish.datamodel.flow.Flow]*)
A single hop in a flow trace.

Variables

- **edge** – The `Edge` identifying the hop/link
- ***routes*** – The routes which caused this hop
- **transformedFlow** – The transformed version of the flow (if NAT is present)

class pybatfish.datamodel.flow.**HeaderConstraints**(*srcIps: Optional[str] = None*, *dstIps: Optional[str] = None*, *srcPorts=None*, *dstPorts=None*, *ipProtocols=None*, *applications=None*, *icmpCodes=None*, *icmpTypes=None*, *firewallClassifications=None*, *ecns=None*, *dscps=None*, *packetLengths=None*, *fragmentOffsets=None*, *tcpFlags: Optional[pybatfish.datamodel.flow.MatchTcpFlags] = None*)
Constraints on an IPv4 packet header space.

Specify constraints on packet headers by specifying lists of allowed values in each field of IP packet.

Variables

- **srcIps** (*str*) – Source location/IP
- **dstIps** (*str*) – Destination location/IP
- **srcPorts** – Source ports as list of ranges (e.g., `"22,53-99"`)
- **dstPorts** – Destination ports as list of ranges, (e.g., `"22,53-99"`)
- **applications** – Shorthands for application protocols (e.g., `SSH`, `DNS`, `SNMP`)
- **ipProtocols** – List of well-known IP protocols (e.g., `TCP`, `UDP`, `ICMP`)
- **icmpCodes** – List of integer ICMP codes
- **icmpTypes** – List of integer ICMP types
- **firewallClassifications** – List of flow states as classified by a stateful firewall (e.g., “new”, “established”)
- **dscps** – List of allowed DSCP value ranges
- **ecns** – List of allowed ECN values ranges
- **packetLengths** – List of allowed packet length value ranges

- **fragmentOffsets** – List of allowed fragmentOffset value ranges
- **tcpFlags** – List of *MatchTcpFlags* – conditions on which TCP flags to match

Lists of values in each fields are subject to a logical "OR":

```
>>> HeaderConstraints(ipProtocols=["TCP", "UDP"])
HeaderConstraints(srcIps=None, dstIps=None, srcPorts=None, dstPorts=None,␣
↪ipProtocols=['TCP', 'UDP'], applications=None,
icmpCodes=None, icmpTypes=None, firewallClassifications=None, ecns=None,␣
↪dscps=None, packetLengths=None, fragmentOffsets=None, tcpFlags=None)
```

means allow TCP OR UDP.

Different fields are ANDed together:

```
>>> HeaderConstraints(srcIps="1.1.1.1", dstIps="2.2.2.2", applications=["SSH"])
HeaderConstraints(srcIps='1.1.1.1', dstIps='2.2.2.2', srcPorts=None,␣
↪dstPorts=None, ipProtocols=None, applications=['SSH'],
icmpCodes=None, icmpTypes=None, firewallClassifications=None, ecns=None,␣
↪dscps=None, packetLengths=None, fragmentOffsets=None, tcpFlags=None)
```

means an SSH connection originating at `1.1.1.1` and going to `2.2.2.2`

Any `None` values will be treated as unconstrained.

class pybatfish.datamodel.flow.**Hop**(*node: str, steps: List[pybatfish.datamodel.flow.Step]*)
A single hop in a flow trace.

Variables

- **node** – Name of node considered as the Hop
- **steps** – List of steps taken at this Hop

class pybatfish.datamodel.flow.**InboundStepDetail**
Details of a step representing the receiving (acceptance) of a flow into a Hop.

class pybatfish.datamodel.flow.**MatchSessionStepDetail**
Details of a step for when a flow matches a firewall session.

class pybatfish.datamodel.flow.**MatchTcpFlags**(*tcpFlags: pybatfish.datamodel.flow.TcpFlags, useAck: bool = True, useCwr: bool = True, useEce: bool = True, useFin: bool = True, usePsh: bool = True, useRst: bool = True, useSyn: bool = True, useUrg: bool = True*)
Match given *TcpFlags*.

For each bit in the TCP flags, a *useX* must be set to true, otherwise the bit is treated as "don't care".

Variables

- **tcpFlags** – tcp flags to match
- **useAck** –
- **useCwr** –
- **useEce** –
- **useFin** –
- **usePsh** –

- **useRst** –
- **useSyn** –
- **useUrg** –

static match_ack()
Return match conditions checking that ACK bit is set.

static match_established()
Return a list of match conditions matching an established flow (ACK or RST bit set).

static match_rst()
Return match conditions checking that RST bit is set.

class pybatfish.datamodel.flow.**OriginateStepDetail**(*originatingVrf: str*)
Details of a step representing the originating of a flow in a Hop.

Variables **originatingVrf** – VRF from which the Flow originates

class pybatfish.datamodel.flow.**RoutingStepDetail**(*routes: List[Any]*)
Details of a step representing the routing from input interface to output interface.

Variables ***routes*** – List of routes which were considered to select the output interface

class pybatfish.datamodel.flow.**SetupSessionStepDetail**
Details of a step for when a firewall session is created.

class pybatfish.datamodel.flow.**PathConstraints**(*startLocation: Optional[str] = None*, *endLocation: Optional[str] = None*, *transitLocations: Optional[str] = None*, *forbiddenLocations: Optional[str] = None*)
Constraints on the path of a flow.

Variables

- **startLocation** – Location description of where a flow is allowed to start
- **endLocation** – Location description of where a flow is allowed to terminate
- **transitLocation** – Location description of where a flow must transit
- **startLocation** – Location description of where a flow is *not* allowed to transit

class pybatfish.datamodel.flow.**TcpFlags**(*ack: bool = False*, *cwr: bool = False*, *ece: bool = False*, *fin: bool = False*, *psh: bool = False*, *rst: bool = False*, *syn: bool = False*, *urg: bool = False*)
Represents a set of TCP flags in a packet.

Variables

- **ack** –
- **cwr** –
- **ece** –
- **fin** –
- **psh** –
- **rst** –
- **syn** –
- **urg** –

class pybatfish.datamodel.flow.**Trace**(*disposition: str, hops: List[pybatfish.datamodel.flow.Hop]*)

A trace of a flow through the network.

A Trace is a combination of hops and flow fate (i.e., disposition).

Variables

- **disposition** – Flow disposition
- **hops** – A list of hops (*Hop*) the flow took

class pybatfish.datamodel.flow.**TransformationStepDetail**(*transformationType: str, flowDiffs: List[pybatfish.datamodel.flow.FlowDiff]*)

Details of a step representation a packet transformation.

Variables

- **transformationType** – The type of the transformation
- **flowDiffs** – Set of changed flow fields

3.4 Reference Library

class pybatfish.datamodel.referencelibrary.**AddressGroup**(*name: str, addresses: List[str] = NOTHING*)

Information about an address group.

Variables

- **name** – The name of the group
- **addresses** – a list of 'addresses' where each element is a string that represents an IP address (e.g., "1.1.1.1") or an address:mask (e.g., "1.1.1.1:0.0.0.8").

class pybatfish.datamodel.referencelibrary.**InterfaceGroup**(*name: str, interfaces: List[pybatfish.datamodel.primitives.Interface] = NOTHING*)

Information about an interface group.

Variables

- **name** – The name of the group
- **interfaces** – a list of interfaces, of type `Interface`.

class pybatfish.datamodel.referencelibrary.**NodeRole**(*name: str, regex: str*)

Information about a node role.

Variables

- **name** – Name of the node role.
- **regex** – A regular expression over node names to describe nodes that belong to this role. The regular expression must be a valid **Java** regex.

class pybatfish.datamodel.referencelibrary.**NodeRoleDimension**(*name: str, type: str = 'CUSTOM', roles: List[pybatfish.datamodel.referencelibrary.N = NOTHING*)

Information about a node role dimension.

Variables

- **name** – Name of the node role dimension.
- **type** – to capture if the dimension contains automatically inferred roles (AUTO) or user-defined roles (CUSTOM).
- **roles** – The list of *NodeRole* objects in this dimension.

class pybatfish.datamodel.referencelibrary.**NodeRolesData**(*roleDimensions: List[pybatfish.datamodel.referencelibrary.NodeRo = NOTHING*)

Information about a node roles data.

Variables **roleDimensions** – A list of *NodeRoleDimension* objects

class pybatfish.datamodel.referencelibrary.**ReferenceBook**(*name: str*, *addressGroups: List[pybatfish.datamodel.referencelibrary.Address = NOTHING*, *interfaceGroups: List[pybatfish.datamodel.referencelibrary.Interfac = NOTHING*)

Information about a reference book.

Variables

- **name** – Name of the reference book.
- **addressGroups** – A list of groups, of type *AddressGroup*.
- **interfaceGroups** – A list of groups, of type *InterfaceGroup*.

class pybatfish.datamodel.referencelibrary.**ReferenceLibrary**(*books: List[pybatfish.datamodel.referencelibrary.Ref = NOTHING*)

Information about a reference library.

Variables **books** – A list of books of type *ReferenceBook*.

3.5 Answers

class pybatfish.datamodel.answer.base.**Answer**

Represents a generic Batfish answer.

dict()

A dictionary representation of the full answer.

question_name()

Return the name of the question that produced this answer.

class pybatfish.datamodel.answer.table.**TableAnswer**(*dictionary*)

Batfish answer in the form of a table.

excluded_frame(*exclusion_name*)

Return the excluded data for exclusion_name as a pandas.DataFrame.

frame()

Return answer data as a pandas.DataFrame.

CHAPTER 4

Assertion helpers

Utility assert functions for writing network tests (or policies).

All *assert_** methods will raise an `BatfishAssertException` if the assertion fails.

`pybatfish.client.asserts.`**`assert_dict_match`**(*actual*, *expected*, *soft=False*)

Assert that two dictionaries are equal. *expected* can be a subset of *actual*.

Parameters

- **`actual`** – the value tested.
- **`expected`** – the expected value of a dictionary
- **`soft`** – whether this assertion is soft (i.e., generates a warning but not a failure)

`pybatfish.client.asserts.`**`assert_filter_denies`**(*filter_name*, *headers*, *startLocation=None*, *soft=False*)

Check if a named ACL denies a specified set of flows.

Parameters

- **`filter_name`** – the name of ACL to check
- **`headers`** – *`HeaderConstraints`*
- **`startLocation`** – LocationSpec indicating where a flow starts
- **`soft`** – whether this assertion is soft (i.e., generates a warning but not a failure)

Returns True if the assertion passes

`pybatfish.client.asserts.`**`assert_filter_permits`**(*filter_name*, *headers*, *startLocation=None*, *soft=False*)

Check if a named ACL permits a specified set of flows.

Parameters

- **`filter_name`** – the name of ACL to check
- **`headers`** – *`HeaderConstraints`*

- **startLocation** – LocationSpec indicating where a flow starts
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)

Returns True if the assertion passes

pybatfish.client.asserts.**assert_has_no_route**(*routes*, *expected_route*, *node*, *vrf='default'*, *soft=False*)

Assert that a particular route is **NOT** present.

Note: If a node or VRF is missing in the route answer the assertion will NOT fail, but a warning will be generated.

Parameters

- **routes** – All routes returned by the Batfish routes question.
- **expected_route** – A dictionary describing route to match.
- **node** – node hostname on which to look for expected route.
- **vrf** – VRF name to check. Default is *default*.
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

pybatfish.client.asserts.**assert_has_route**(*routes*, *expected_route*, *node*, *vrf='default'*, *soft=False*)

Assert that a particular route is present.

Parameters

- **routes** – All routes returned by the Batfish routes question.
- **expected_route** – A dictionary describing route to match.
- **node** – node hostname on which to look for a route.
- **vrf** – VRF name where the route should be present. Default is *default*.
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

pybatfish.client.asserts.**assert_num_results**(*answer*, *num*, *soft=False*)

Assert an exact number of results were returned.

Parameters

- **answer** – Batfish answer or DataFrame
- **num** (*int*) – expected number of results
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

pybatfish.client.asserts.**assert_zero_results**(*answer*, *soft=False*)

Assert no results were returned.

Parameters

- **answer** – Batfish answer or DataFrame
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

CHAPTER 5

API reference

5.1 Client API

Contains Batfish client commands that query the Batfish service.

pybatfish.client.commands.**bf_add_issue_config**(*issue_config*)

Add or update the active network's configuration for an issue .

Parameters **issue_config** (pybatfish.settings.issues.IssueConfig) – The IssueConfig object to add or update

pybatfish.client.commands.**bf_auto_complete**(*completion_type*, *query*, *max_suggestions=None*)

Get a list of autocomplete suggestions that match the provided query based on the variable type.

If completion is not supported for the provied variable type a BatfishException will be raised.

Usage Example:

```
>>> from pybatfish.client.commands import bf_auto_complete, bf_set_network
>>> from pybatfish.datamodel.primitives import AutoCompleteSuggestion,␣
↪VariableType
>>> name = bf_set_network()
>>> bf_auto_complete(VariableType.ROUTING_PROTOCOL_SPEC, "b")
[AutoCompleteSuggestion(description=None, insertion_index=0, is_partial=False,␣
↪rank=2147483647, text='bgp'),
    AutoCompleteSuggestion(description=None, insertion_index=0, is_partial=False,␣
↪rank=2147483647, text='ebgp'),
    AutoCompleteSuggestion(description=None, insertion_index=0, is_partial=False,␣
↪rank=2147483647, text='ibgp')]
```

Parameters

- **completion_type** (*VariableType*) – The type of parameter to suggest autocompletions for
- **query** (*str*) – The partial string to match suggestions on

- **max_suggestions** (*int*) – Optional max number of suggestions to be returned

pybatfish.client.commands.**bf_delete_issue_config**(*major*, *minor*)
Deletes the issue config for the active network.

pybatfish.client.commands.**bf_delete_network**(*name*)
Delete network by name.

Parameters **name** (*string*) – name of the network to delete

pybatfish.client.commands.**bf_delete_node_role_dimension**(*dimension*)
Deletes the definition of the given role dimension for the active network.

pybatfish.client.commands.**bf_delete_reference_book**(*book_name*)
Deletes the reference book with the specified name for the active network.

pybatfish.client.commands.**bf_delete_snapshot**(*name*)
Delete named snapshot from current network.

Parameters **name** (*string*) – name of the snapshot to delete

pybatfish.client.commands.**bf_extract_answer_summary**(*answer_dict*)
Get the answer for a previously asked question.

pybatfish.client.commands.**bf_fork_snapshot**(*base_name*, *name=None*, *overwrite=False*, *background=False*, *deactivate_interfaces=None*, *deactivate_links=None*, *deactivate_nodes=None*, *restore_interfaces=None*, *restore_links=None*, *restore_nodes=None*, *add_files=None*, *extra_args=None*)
Copy an existing snapshot and deactivate or reactivate specified interfaces, nodes, and links on the copy.

Parameters

- **base_name** (*string*) – name of the snapshot to copy
- **name** (*string*) – name of the snapshot to initialize
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name
- **background** (*bool*) – whether or not to run the task in the background
- **deactivate_interfaces** (*list[Interface]*) – list of interfaces to deactivate in new snapshot
- **deactivate_links** (*list[Edge]*) – list of links to deactivate in new snapshot
- **deactivate_nodes** (*list[str]*) – list of names of nodes to deactivate in new snapshot
- **restore_interfaces** (*list[Interface]*) – list of interfaces to reactivate
- **restore_links** (*list[Edge]*) – list of links to reactivate
- **restore_nodes** (*list[str]*) – list of names of nodes to reactivate
- **add_files** (*str*) – path to zip file or directory containing files to add
- **extra_args** (*dict*) – extra arguments to be passed to the parse command. See bf_session.additionalArgs.

Returns name of initialized snapshot, JSON dictionary of task status if background=True, or None if the call fails

Return type Union[str, Dict, None]

pybatfish.client.commands.**bf_generate_dataplane**(*snapshot=None*, *extra_args=None*)
Generates the data plane for the supplied snapshot. If no snapshot argument is given, uses the last snapshot initialized.

pybatfish.client.commands.**bf_get_analysis_answers**(*name*, *snapshot=None*, *reference_snapshot=None*)
Get the answers for a previously asked analysis.

pybatfish.client.commands.**bf_get_answer**(*questionName*, *snapshot*, *reference_snapshot=None*)
Get the answer for a previously asked question.

Parameters

- **questionName** – the unique identifier of the previously asked question
- **snapshot** – the snapshot the question is run on
- **reference_snapshot** – if present, the snapshot against which the answer was computed differentially.

pybatfish.client.commands.**bf_get_issue_config**(*major*, *minor*)
Returns the issue config for the active network.

pybatfish.client.commands.**bf_get_node_role_dimension**(*dimension*)
Returns the definition of the given node role dimension for the active network.

pybatfish.client.commands.**bf_get_node_roles**()
Returns the definitions of node roles for the active network.

pybatfish.client.commands.**bf_get_reference_book**(*book_name*)
Returns the reference book with the specified for the active network.

pybatfish.client.commands.**bf_get_reference_library**()
Returns the reference library for the active network.

pybatfish.client.commands.**bf_get_snapshot_inferred_node_role_dimension**(*dimension*)
Gets the suggested definition and hypothetical assignments of node roles for the given inferred dimension for the active network and snapshot.

pybatfish.client.commands.**bf_get_snapshot_inferred_node_roles**()
Gets suggested definitions and hypothetical assignments of node roles for the active network and snapshot.

pybatfish.client.commands.**bf_get_snapshot_node_role_dimension**(*dimension*)
Returns the defintion and assignments of node roles for the given dimension for the active network and snapshot.

pybatfish.client.commands.**bf_get_snapshot_node_roles**()
Returns the definitions and assignments of node roles for the active network and snapshot.

pybatfish.client.commands.**bf_init_snapshot**(*upload*, *name=None*, *overwrite=False*, *background=False*, *extra_args=None*)
Initialize a new snapshot.

Parameters

- **upload** (*zip file or directory*) – snapshot to upload
- **name** (*string*) – name of the snapshot to initialize
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name
- **background** (*bool*) – whether or not to run the task in the background
- **extra_args** (*dict*) – extra arguments to be passed to the parse command. See bf_session.additionalArgs.

Returns name of initialized snapshot, or JSON dictionary of task status if background=True

Return type Union[str, Dict]

pybatfish.client.commands.**bf_list_networks**()
List networks the session's API key can access.

Returns a list of network names

pybatfish.client.commands.**bf_list_snapshots**(*verbose=False*)
List snapshots for the current network.

Parameters **verbose** – If true, return the full output of Batfish, including snapshot metadata.

Returns a list of snapshot names or the full json response containing snapshots and metadata (if *verbose=True*)

pybatfish.client.commands.**bf_put_node_role_dimension**(*dimension*)
Put a role dimension in the active network.

Overwrites the old dimension if one of the same name already exists.

Individual roles within the dimension must have a valid (java) regex. The node list within those roles, if present, is ignored by the server.

Parameters **dimension** (*pybatfish.datamodel.referencelibrary.NodeRoleDimension*) – The NodeRoleDimension object for the dimension to add

pybatfish.client.commands.**bf_put_node_roles**(*node_roles_data*)
Writes the definitions of node roles for the active network. Completely replaces any existing definitions.

pybatfish.client.commands.**bf_read_question_settings**(*question_class*, *json_path=None*)
Retrieves the network-wide JSON settings tree for the specified question class.

Parameters

- **question_class** (*string*) – The class of question whose settings are to be read
- **json_path** (*list*) – If supplied, return only the subtree reached by successively traversing each key in json_path starting from the root.

pybatfish.client.commands.**bf_put_reference_book**(*book*)
Put a reference book in the active network.

If a book with the same name exists, it is overwritten.

Parameters **book** (*pybatfish.datamodel.referencelibrary.ReferenceBook*) – The ReferenceBook object to add

pybatfish.client.commands.**bf_set_network**(*name=None*, *prefix='pcp'*)
Configure the network used for analysis.

Parameters

- **name** (*string*) – name of the network to set. If *None*, a name will be generated using prefix.
- **prefix** – prefix to prepend to auto-generated network names if name is empty

Returns The name of the configured network, if configured successfully.

Return type string

Raises **BatfishException** – if configuration fails

pybatfish.client.commands.**bf_set_snapshot**(*name=None*, *index=None*)
Set the current snapshot by name or index.

Parameters

- **name** (*string*) – name of the snapshot to set as the current snapshot
- **index** (*int*) – set the current snapshot to the `index`-th most recent snapshot

Returns the name of the successfully set snapshot

Return type str

pybatfish.client.commands.**bf_upload_diagnostics**(*dry_run=True*, *netconan_config=None*)
Fetch, anonymize, and optionally upload snapshot diagnostics information.

This runs a series of diagnostic questions on the current snapshot (including collecting parsing and conversion information).

The information collected is anonymized with Netconan which either anonymizes passwords and IP addresses (default) or uses the settings in the provided *netconan_config*.

The anonymous information is then either saved locally (if *dry_run* is True) or uploaded to Batfish developers (if *dry_run* is False). The uploaded information will be accessible only to Batfish developers and will be used to help diagnose any issues you encounter.

Parameters

- **dry_run** (*bool*) – whether or not to skip upload; if False, anonymized files will be stored locally, otherwise anonymized files will be uploaded to Batfish developers
- **netconan_config** (*string*) – path to Netconan configuration file

Returns location of anonymized files (local directory if doing dry run, otherwise upload ID)

Return type string

pybatfish.client.commands.**bf_write_question_settings**(*settings*, *question_class*, *json_path=None*)
Write the network-wide JSON settings tree for the specified question class.

Parameters

- **settings** (*dict*) – The JSON representation of the settings to be written
- **question_class** (*string*) – The class of question to configure
- **json_path** (*list*) – If supplied, write settings to the subtree reached by successively traversing each key in json_path starting from the root. Any absent keys along the path will be created.

5.2 Session parameters

class pybatfish.client.session.**Session**(*logger*)
Keeps session configuration needed to connect to a Batfish server.

Variables

- **coordinatorHost** – The host of the batfish service
- **coordinatorPort** – The port batfish service is running on (9997 by default)
- **coordinatorPort2** – The additional port of batfish service (9996 by default)

- **useSsl** – Whether to use SSL when connecting to Batfish (False by default)
- **apiKey** – Your API key

get_base_url()
Generate the base URL for connecting to batfish coordinator.

get_base_url2()
Generate the base URL for V2 of the coordinator APIs.

5.3 Question API

class pybatfish.question.question.**QuestionBase**(*dictionary*)
All questions inherit functionality from this class.

answer(*snapshot=None*, *reference_snapshot=None*, *include_one_table_keys=None*, *background=False*, *extra_args=None*)
Ask and return the answer for this question.

Parameters

- **snapshot** (*str*) – the snapshot on which to answer the question. If not provided, the latest snapshot initialized will be used.
- **reference_snapshot** (*str*) – for differential questions only, the snapshot against which to compare.
- **include_one_table_keys** (*bool*) – if differential is True, include keys only from one table and not both.
- **background** (*bool*) – run this question in background, return immediately
- **extra_args** (*dict*) – extra arguments to be passed to the parse command. See bf_session.additionalArgs.

Return type *Answer* or *TableAnswer*

Raises **QuestionValidationException** – if the question is malformed

dict()
Return the dictionary representing this question.

get_description()
Return the short description of this question.

get_differential()
Return whether this question is to be asked differentially.

get_include_one_table_keys()
Return whether keys present in only one table should be included when computing answer table diffs.

get_long_description()
Return the long description of this question.

get_name()
Return the name of this question.

json(***kwargs*)
Return the json string representing this question.

Keyword arguments passed to json.dumps with default assignments of sort_keys=True and indent=2

make_check()
Make this question a check which asserts that there are no results.

set_assertion(*assertion*)
Set an assertion for a given question.

Overwrites any previous assertions.

Defines Batfish questions and logic for loading them from disk or Batfish.

pybatfish.question.question.**list_questions**(*tags=None*, *question_module='pybatfish.question.bfq'*)
List available questions.

Parameters

- **tags** – if not *None*, only list questions with given tags. See *list_tags()* for a list of tags given currently loaded questions.
- **question_module** – which module to load the questions from. By default, *pybatfish.question.bfq* is used.

Returns a list of questions, where each question is represented as a dict containing "name", "description", and "tags".

pybatfish.question.question.**list_tags**()
List tags across all available questions.

pybatfish.question.question.**load_dir_questions**(*questionDir*, *moduleName='pybatfish.question.bfq'*)
Load question templates from a directory on disk and install them in the given module.

pybatfish.question.question.**load_questions**(*question_dir=None*, *from_server=False*, *module_name='pybatfish.question.bfq'*)
Load questions from directory or batfish service.

Parameters

- **question_dir** (*str*) – Load questions from this local directory instead of remote questions from the batfish service.
- **from_server** (*bool*) – if true or *question_dir* is None, load questions from service.
- **module_name** – the name of the module where questions should be loaded. Default is *pybatfish.question.bfq*

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

Z