priority

Release 2.0.0

Jun 27, 2021

Contents

1 Installation **3**

2 Using Priority **5**
- 2.1 Iterating The Tree . . . 5
- 2.2 Updating The Tree . . . 6
- 2.3 Removing Streams . . . 6

3 Priority API **7**
- 3.1 Priority Tree . . . 7
- 3.2 Exceptions . . . 8

4 Vulnerability Notifications **9**
- 4.1 Known Vulnerabilities . . . 9

5 License **11**

6 Contributors **13**
- 6.1 Development Lead . . . 13
- 6.2 Contributors . . . 13

Index **15**

Priority is a pure-Python implementation of the priority logic for HTTP/2, set out in RFC 7540 Section 5.3 (Stream Priority). This logic allows for clients to express a preference for how the server allocates its (limited) resources to the many outstanding HTTP requests that may be running over a single HTTP/2 connection.

Specifically, this Python implementation uses a variant of the implementation used in the excellent H2O project. This original implementation is also the inspiration for nghttp2's priority implementation, and generally produces a very clean and even priority stream. The only notable changes from H2O's implementation are small modifications to allow the priority implementation to work cleanly as a separate implementation, rather than being embedded in a HTTP/2 stack directly.

Contents:

CHAPTER 1

Installation

Priority is a pure-Python project. This means installing it is extremely simple. To get the latest release from PyPI, simply run:

```
$ pip install priority
```

CHAPTER 2

Using Priority

Priority has a simple API. Streams are inserted into the tree: when they are inserted, they may optionally have a weight, depend on another stream, or become an exclusive dependent of another stream. To manipulate the tree, we use a *PriorityTree* object.

```
>>> p = priority.PriorityTree()
>>> p.insert_stream(stream_id=1)
>>> p.insert_stream(stream_id=3)
>>> p.insert_stream(stream_id=5, depends_on=1)
>>> p.insert_stream(stream_id=7, weight=32)
>>> p.insert_stream(stream_id=9, depends_on=7, weight=8)
>>> p.insert_stream(stream_id=11, depends_on=7, exclusive=True)
```

Once streams are inserted, the stream priorities can be requested. This allows the server to make decisions about how to allocate resources.

2.1 Iterating The Tree

The tree in this algorithm acts as a gate. Its goal is to allow one stream "through" at a time, in such a manner that all the active streams are served as evenly as possible in proportion to their weights.

This is handled in Priority by iterating over the tree. The tree itself is an iterator, and each time it is advanced it will yield a stream ID. This is the ID of the stream that should next send data.

This looks like this:

```
>>> for stream_id in p:
...     send_data(stream_id)
```

If each stream only sends when it is 'ungated' by this mechanism, the server will automatically be emitting stream data in conformance to RFC 7540.

2.2 Updating The Tree

If for any reason a stream is unable to proceed (for example, it is blocked on HTTP/2 flow control, or it is waiting for more data from another service), that stream is *blocked*. The *PriorityTree* should be informed that the stream is blocked so that other dependent streams get a chance to proceed. This can be done by calling the *block* method of the tree with the stream ID that is currently unable to proceed. This will automatically update the tree, and it will adjust on the fly to correctly allow any streams that were dependent on the blocked one to progress.

For example:

```
>>> for stream_id in p:
...     send_data(stream_id)
...     if blocked(stream_id):
...         p.block(stream_id)
```

When a stream goes from being blocked to being unblocked, call the *unblock* method to place it back into the sequence. Both the *block* and *unblock* methods are idempotent and safe to call repeatedly.

Additionally, the priority of a stream may change. When it does, the *reprioritize* method can be used to update the tree in the wake of that change. *reprioritize* has the same signature as *insert_stream*, but applies only to streams already in the tree.

2.3 Removing Streams

A stream can be entirely removed from the tree by calling *remove_stream*. Note that this is *not* idempotent. Further, calling *remove_stream* and then re-adding it *may* cause a substantial change in the shape of the priority tree, and *will* cause the iteration order to change.

CHAPTER 3

Priority API

3.1 Priority Tree

class priority.**PriorityTree**(*maximum_streams: int = 1000*)
A HTTP/2 Priority Tree.

This tree stores HTTP/2 streams according to their HTTP/2 priorities.

Changed in version 1.2.0: Added `maximum_streams` keyword argument.

Parameters **maximum_streams** (`int`) – The maximum number of streams that may be active in the priority tree at any one time. If this number is exceeded, the priority tree will raise a *TooManyStreamsError* and will refuse to insert the stream.

This parameter exists to defend against the possibility of DoS attack by attempting to overfill the priority tree. If any endpoint is attempting to manage the priority of this many streams at once it is probably trying to screw with you, so it is sensible to simply refuse to play ball at that point.

While we allow the user to configure this, we don't really *expect* them too, unless they want to be even more conservative than we are by default.

block(*stream_id: int*) → None
Marks a given stream as blocked, with no data to send.

Parameters **stream_id** – The ID of the stream to block.

insert_stream(*stream_id: int*, *depends_on: Optional[int] = None*, *weight: int = 16*, *exclusive: bool = False*) → None
Insert a stream into the tree.

Parameters

- **stream_id** – The stream ID of the stream being inserted.
- **depends_on** – (optional) The ID of the stream that the new stream depends on, if any.
- **weight** – (optional) The weight to give the new stream. Defaults to 16.

- **exclusive** – (optional) Whether this new stream should be an exclusive dependency of the parent.

remove_stream(*stream_id: int*) → None
Removes a stream from the priority tree.

Parameters **stream_id** – The ID of the stream to remove.

reprioritize(*stream_id: int*, *depends_on: Optional[int] = None*, *weight: int = 16*, *exclusive: bool = False*) → None
Update the priority status of a stream already in the tree.

Parameters

- **stream_id** – The stream ID of the stream being updated.
- **depends_on** – (optional) The ID of the stream that the stream now depends on. If `None`, will be moved to depend on stream 0.
- **weight** – (optional) The new weight to give the stream. Defaults to 16.
- **exclusive** – (optional) Whether this stream should now be an exclusive dependency of the new parent.

unblock(*stream_id: int*) → None
Marks a given stream as unblocked, with more data to send.

Parameters **stream_id** – The ID of the stream to unblock.

3.2 Exceptions

class priority.**PriorityError**
The base class for all `priority` exceptions.

class priority.**DeadlockError**
Raised when there are no streams that can make progress: all streams are blocked.

class priority.**PriorityLoop**
An unexpected priority loop has been detected. The tree is invalid.

class priority.**DuplicateStreamError**
An attempt was made to insert a stream that already exists.

class priority.**MissingStreamError**
An operation was attempted on a stream that is not present in the tree.

class priority.**TooManyStreamsError**
An attempt was made to insert a dangerous number of streams into the priority tree at the same time.

New in version 1.2.0.

class priority.**BadWeightError**
An attempt was made to create a stream with an invalid weight.

New in version 1.3.0.

class priority.**PseudoStreamError**
An operation was attempted on stream 0.

New in version 1.3.0.

CHAPTER 4

Vulnerability Notifications

This section of the page contains all known vulnerabilities in the priority library. These vulnerabilities have all been reported to us via our vulnerability disclosure policy.

4.1 Known Vulnerabilities

#	Vulnerability	Date Announced	First Version	Last Version	CVE
1	DoS via unlimited stream insertion.	2016-08-04	1.0.0	1.1.1	CVE-2016-6580

CHAPTER 5

License

Copyright (c) 2015 Cory Benfield

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 6

Contributors

Priority is written and maintained by Cory Benfield and various contributors:

6.1 Development Lead

- Cory Benfield <cory@lukasa.co.uk>

6.2 Contributors

In chronological order:

Index

B

BadWeightError (*class in priority*), 8
block() (*priority.PriorityTree method*), 7

D

DeadlockError (*class in priority*), 8
DuplicateStreamError (*class in priority*), 8

I

insert_stream() (*priority.PriorityTree method*), 7

M

MissingStreamError (*class in priority*), 8

P

PriorityError (*class in priority*), 8
PriorityLoop (*class in priority*), 8
PriorityTree (*class in priority*), 7
PseudoStreamError (*class in priority*), 8

R

remove_stream() (*priority.PriorityTree method*), 8
reprioritize() (*priority.PriorityTree method*), 8

T

TooManyStreamsError (*class in priority*), 8

U

unblock() (*priority.PriorityTree method*), 8