multicast Documentation

Release 0.1.1alpha1

Iain Lowe

September 07, 2013

CONTENTS

CHAPTER

ONE

BASIC ARCHITECTURE

This module provides classes and functions for asynchronous multicast UDP clients and servers.

The *multicast* module has a default `Poller` object in the *multicast._defaultPoller* attribute. The convenience funtions the module exposes use the default `Poller` instance.

By default, instances of *multicast* classes register themselves with the default `Poller` object. To implement custom polling setups, instantiate other `Poller` objects and manually *register* the `Multicaster` and `MulticastListener` instances.

CHAPTER

TWO

MULTICASTING

2.1 Simple multicast

The simplest way to send a packet to a multicast group is by using the `multicast.sendto()` function.

`multicast.`**`sendto`**(*packets*, *ip='224.0.42.42'*, *port=4242*, *ttl=1*)

Send *packets* to the multicast group at *ip* on *port*.

The packet is sent with a TTL of 1 by default; see below for suggested TTL values:

same host	0
same subnet	1
same site	32
same region	64
same continent	128
unrestricted	255

2.2 Multicasters

class `multicast.`**`Multicaster`**(*ip='224.0.42.42'*, *port=4242*, *ttl=1*)

Make sub-classes of this class iterable and yield packets to multicast.

The TTL of packets sent by the multicaster defaults to 1 (ie. the local segment). See the table below for suggested TTL values.

same host	0
same subnet	1
same site	32
same region	64
same continent	128
unrestricted	255

CHAPTER

THREE

MULTICAST LISTENERS

3.1 Simple multicast monitoring

multicast.**listen**(*packet_handler*, *ip='224.0.42.42'*, *port=4242*, *mtu=4096*, *limit=None*)

Calls a *packet_handler* with a *packet* and an *addr* argument each time a packet is received by the multicast group at *ip* on *port*.

Note: This function does not return until the *packet_handler* returns non-*False*

---

mtu is the maximum packet size in bytes.

```
>>> from multicast import listen, sendto, poll
>>> listener = listen()
>>> sendto('test')
>>> poll()
>>> listener.next()
'test'
```

Warning: Clients of this function **must** call `multicast.poll()` after each call to the returned generator's `next()` method.
Failure to do so will result in no packets being delivered.

3.2 Multicast listener

class multicast.**MulticastListener**(*filter=None*, *ip='224.0.42.42'*, *port=4242*, *mtu=4096*)

MulticastListeners join a multicast IP group and call their `handle_packet()` method when packets are sent to the group.

Packets are read up to *mtu* bytes of each packet, and only listen for packets that match *filter*.

filter can be a callable **filter(packet, addr)** that returns *True* if the packet should be accepted or *False* to drop the packet; or it can be a regular expression, in which case it is compiled and packets that match it are accepted.

handle_packet(*packet*, *addr*)

Implemented by sub-classes as a callback when packets matching the *filter* are received.

3.3 Example of multicast listener

Listen for a single request and quit:

```
>>> from datetime import datetime
>>> from multicast import sendto, poll, MulticastListener
>>> class Printer(MulticastListener):
...     filter = lambda packet, addr: True
...     def handle_packet(self, packet, addr):
...             print packet
...             return True # Stop listening
...
>>> Printer()
>>> sendto('test')
>>> poll()
'testa'
```

CHAPTER

FOUR

EPOLL OBJECT POLLING

4.1 Simple polling

The *multicast* module provides four convenience functions that mirror the methods on `Poller` instances. These functions operate on the default `Poller` instance and provide a simple way to control polling.

Note: References to *self* in this section refer to the default `Poller`.

`multicast.`**`register`**(*self*, *fd*, *flags=0*)
Register a file descriptor object with the `Poller`. Future calls to `poll()` will check whether the file descriptor has any pending I/O events. *fd* must be an object that implements a **fileno()** method that returns an integer. It must also support implement the following methods:

handle_read() To indicate that it wants to receive EPOLLIN and EPOLLPRI events

handle_write() To indicate that it wants to receive EPOLLOUT events

`multicast.`**`unregister`**(*self*, *fd*)
Remove a tracked file descriptor

`multicast.`**`poll`**(*self*, *timeout=0*)
Polls the registered set of file descriptors for events.

When an event is detected on a file descriptor, the corresponding **handle_<event>()** method is called on the registered object.

`multicast.`**`loop`**(*self*, *timeout=0*, *interval=0*)
Enter a polling loop that terminates when *self.polling* is False.

The *interval* parameter indicates how long to *sleep* between polls. The *timeout* parameter is passed to the *poll* call. Both are expressed in seconds. *interval* and *timeout* both default to **0**.

4.2 Poller objects

For more fine-grained control, or for implementing multiple polling loops, multiple instances of the `Poller` class can be created.

class `multicast.`**`Poller`**
Poller objects wrap `select.epoll()` epoll objects.

They implement a `loop()` function that loops and sleeps polling the underlying *epoll* object for events on registered file descriptors.

loop(*timeout=0*, *interval=0*)
Enter a polling loop that terminates when *self.polling* is False.

The *interval* parameter indicates how long to *sleep* between polls. The *timeout* parameter is passed to the *poll* call. Both are expressed in seconds. *interval* and *timeout* both default to **0**.

poll(*timeout=0*)
Polls the registered set of file descriptors for events.

When an event is detected on a file descriptor, the corresponding **handle_<event>()** method is called on the registered object.

register(*fd*, *flags=0*)
Register a file descriptor object with the `Poller`. Future calls to `poll()` will check whether the file descriptor has any pending I/O events. *fd* must be an object that implements a **fileno()** method that returns an integer. It must also support implement the following methods:

handle_read() To indicate that it wants to receive EPOLLIN and EPOLLPRI events

handle_write() To indicate that it wants to receive EPOLLOUT events

unregister(*fd*)
Remove a tracked file descriptor

PYTHON MODULE INDEX

m