simple-ostinato

Release 0.0.1

May 23, 2017

Usage

CHAPTER 1

The drone object

We suppose that drone is running on the local host.

Connecting to drone

All the operations are performed from a `simple_ostinato.Drone` object, that holds the connection to a drone agent, and perform all the protocol buffer calls.

```
from simple_ostinato import Drone

# create an instance:
drone = Drone('localhost')
```

When the object is initialized, a connection is established to the drone instance running on *localhost*. This will fail if no drone instance is running, so it is also possible to disable the automatic connection:

```
from simple_ostinato import Drone

# create an instance:
drone = Drone('localhost', connect=False)

# do things...

# later, the connection can be established:
drone.connect()
```

Retrieving the ports

After establishing the connection we can fetch the ports available on the drone instance:

```
drone.fetch_ports()
for port in drone.ports:
    print str(port)
```

Output:

```
veth0 (id=0, enabled=True)
veth1 (id=1, enabled=True)
enp0s25 (id=2, enabled=True)
any (id=3, enabled=True)
lo (id=4, enabled=True)
wlp3s0 (id=5, enabled=True)
docker0 (id=6, enabled=True)
bluetooth0 (id=7, enabled=True)
bluetooth-monitor (id=8, enabled=True)
dbus-system (id=9, enabled=True)
dbus-session (id=10, enabled=True)
```

To get a port by name, we either iterate over the ports or just call `get_port()`:

```
veth0 = drone.get_port('veth0')

# this is equivalent to:
for port in drone.ports:
    if port.name == 'veth0':
        veth0 = port
        break
```

It is also possible to get a port by id with by iterating over the ports, but since I don't really see a use case for getting ports by ID, there is not method for this at the moment:

```
for port in drone.ports:
    if port.port_id == 0:
        veth0 = port
        break
```

Complete example

```
from simple_ostinato import Drone

drone = Drone('localhost')
drone.fetch_ports()

print 'printing all the ports available'
print '--------------------------------'
for port in drone.ports:
    print str(port)
print '--------------------------------'

print '\ngetting port "veth0" by name:'
veth0 = drone.get_port('veth0')

print '\ngetting port with id 1:'
for port in drone.ports:
    if port.port_id == 1:
```

```
        port1 = port
        break
print str(port1)
```

Output:

```
printing all the ports available
--------------------------------
veth0 (id=0, enabled=True)
veth1 (id=1, enabled=True)
enp0s25 (id=2, enabled=True)
any (id=3, enabled=True)
lo (id=4, enabled=True)
wlp3s0 (id=5, enabled=True)
docker0 (id=6, enabled=True)
bluetooth0 (id=7, enabled=True)
bluetooth-monitor (id=8, enabled=True)
dbus-system (id=9, enabled=True)
dbus-session (id=10, enabled=True)
--------------------------------

getting port "veth0" by name:
veth0 (id=0, enabled=True)

getting port with id 1:
veth1 (id=1, enabled=True)
```

CHAPTER 2

Manipulating streams

We suppose that drone is running on the local host.

Creation

To create an empty stream:

```
stream = lo.add_stream()
```

Configuration

At this point, the stream is created on the drone instance, but it has not configuration. To configure it, we simply set the desired attributes on the stream object, and then call `save()` to update the stream configuration on the drone instance.

```
# we give a name to the stream, although it's optional
stream.name = 'a_stream'
# we want to send packets, not bursts of packets
stream.unit = 'PACKETS'
# we want the stream to finish after it sent a fixed amount of packets
stream.mode = 'FIXED'
# after this stream, we want to stop transmitting, even if there is another
# stream enabled on this port
stream.next = 'STOP'
# we want to send 100 packets
stream.num_packets = 100
# at a rate of 50 packets per seconds
stream.packets_per_sec = 50
# finally, we enable the stream, otherwise, it won't send anything.
stream.is_enabled = True
```

The stream object holds the configuration. To apply it we call `save()`:

```
stream.save()
```

Protocols configuration

We will send a simple IP packet with fixed fields. We add the different layers (or protocols) to the stream. Note that the order matters: adding an IPv4 layer before a MAC layer will result in an invalid frame. Each layer correspond to a class in `simple_ostinato.protocols`:

```
from simple_ostinato.protocols import Mac, Ethernet, IPv4, Payload

mac = Mac()
mac.source = '00:00:11:11:22:22'
mac.destination = 'FF:FF:FF:FF:FF:FF'
stream.layers.append(mac)

eth = Ethernet()
eth.ether_type = 0x800
stream.layers.append(eth)

ip = IPv4()
ip.source = '10.0.0.1'
ip.destination = '10.0.0.2'
stream.layers.append(ip)

payload = Payload()
stream.layers.append(payload)

# finally, save the stream
stream.save()
```

Since `simple_ostinato.stream.Stream.layers` is a simple list, we could have writtend directly `stream.layers = [mac, eth, ip, payload]`.

Also, all the attributes of the layers can be passed to the class constructor, so the above could be just written:

```
from simple_ostinato.protocols import Mac, Ethernet, IPv4, Payload

stream.layers = [
    Mac(source='00:00:11:11:22:22', destination='FF:FF:FF:FF:FF:FF'),
    Ethernet(ether_type=0x800),
    IPv4(source='10.0.0.1', destination='10.0.0.2'),
    Payload()]
```

To remove a layer, remove it from the `simple_ostinato.stream.Stream.layers` list, and save the stream. For instance, to delete the `Payload` and `IPv4` layers:

```
del stream.layers[-1]
del stream.layers[-1]
```

Variable fields

Apart from a small number of exceptions, every field can be variable, *i.e.* being increment/decremented/randomized. To control this, each attribute has three other attributes associated:

- `<attribute_name>_mode`: can be `FIXED` (the default), `INCREMENT`, `DECREMENT`, or `RANDOMIZED`

- `<attribute_name>_step`: is an integer that specify by how much the field should be incremented or decremented
- `<attribute_name>_count`: is an integer that specify after how many packets, the field should be reset to its inital value

Some fields also have a `<attribute_name>_override` attribute. This field must be set to `True` in order to set a custom value. Otherwise, Ostinato computes a value automatically.

Deletion

We can delete a stream by id:

```
lo.del_stream(stream.stream_id)
```

Complete example

```
from simple_ostinato import Drone
from simple_ostinato.protocols import Mac, Ethernet, IPv4, Payload

drone = Drone('localhost')
drone.fetch_ports()
lo = drone.get_port('lo')

# create a stream
stream = lo.add_stream()

# configure the stream
stream.name = 'a_stream'
stream.unit = 'PACKETS'
stream.mode = 'FIXED'
stream.next = 'STOP'
stream.num_packets = 100
stream.packets_per_sec = 50
stream.is_enabled = True

# IMPORTANT: apply the configuration
stream.save()

# Add layers
stream.layers = [
    Mac(source='00:00:11:11:22:22', destination='FF:FF:FF:FF:FF:FF'),
    Ethernet(ether_type=0x800),
    IPv4(source='10.0.0.1', destination='10.0.0.2'),
    Payload()]

# Delete the ip and payload layers
stream.layers = stream.layers[:-2]

# Delete stream
lo.del_stream(stream.stream_id)
```

CHAPTER 3

Traffic

Setup

We suppose that drone is running on the local host. We will also use a veth pair than can be created like this:

```
ip link add veth0 type veth peer name veth1
ip link set veth0 up
ip link set veth1 up
```

Steps

We first create a valid stream as shown in the previous section, but on *veth0*, that will be the transmitting port:

```
from simple_ostinato import Drone
from simple_ostinato.protocols import Mac, Ethernet, IPv4, Payload

drone = Drone('localhost')
drone.fetch_ports()

tx_port = drone.get_port('veth0')

stream = tx_port.add_stream()
stream.name = 'a_stream'
stream.unit = 'PACKETS'
stream.mode = 'FIXED'
stream.next = 'STOP'
stream.num_packets = 100
stream.packets_per_sec = 50
stream.is_enabled = True
stream.save()
stream.layers = [
    Mac(source='00:00:11:11:22:22', destination='FF:FF:FF:FF:FF:FF'),
```

```
    Ethernet(ether_type=0x800),
    IPv4(source='10.0.0.1', destination='10.0.0.2'),
    Payload()]
```

veth1 will be capturing traffic:

```
rx_port = drone.get_port('veth1')
```

We first clear the statistics on both ports:

```
rx_port.clear_stats()
tx_port.clear_stats()
```

Sending and capturing is straightforward:

```
import time

rx_port.start_capture()
tx_port.start_send()
time.sleep(3)
tx_port.stop_send()
rx_port.stop_capture()
```

We can get the stats to perform verifications:

```
rx_stats = rx_port.get_stats()
rx_stats = tx_port.get_stats()
```

The capture can also be retrieved as a string, and/or saved as a pcap file:

```
capture_str = rx_port.get_capture()
rx_port.save_capture(capture_str, 'capture.pcap')
```

If you just want to save it, you can directly do:

```
rx_port.get_capture(save_as='capture.pcap')
```

Complete example

```
import time
from simple_ostinato import Drone
from simple_ostinato.protocols import Mac, Ethernet, IPv4, Payload

drone = Drone('localhost')
drone.fetch_ports()

tx_port = drone.get_port('veth0')

stream = tx_port.add_stream()
stream.name = 'a_stream'
stream.unit = 'PACKETS'
stream.mode = 'FIXED'
stream.next = 'STOP'
stream.num_packets = 100
stream.packets_per_sec = 50
```

```
stream.is_enabled = True
stream.save()
stream.layers = [
    Mac(source='00:00:11:11:22:22', destination='FF:FF:FF:FF:FF:FF'),
    Ethernet(ether_type=0x800),
    IPv4(source='10.0.0.1', destination='10.0.0.2'),
    Payload()]

rx_port = drone.get_port('veth1')

rx_port.clear_stats()
tx_port.clear_stats()
rx_port.start_capture()
tx_port.start_send()
time.sleep(3)
tx_port.stop_send()
rx_port.stop_capture()

print 'tx stats:'
pprint.pprint(tx_port.get_stats())
print 'rx stats:'
pprint.pprint(rx_port.get_stats())

print 'saving capture as capture.pcap'
rx_port.get_capture(save_as='capture.pcap')
```

Output:

```
tx stats:
{'rx_bps': 0L,
 'rx_bytes': 0L,
 'rx_bytes_nic': 0,
 'rx_drops': 0L,
 'rx_errors': 0L,
 'rx_fifo_errors': 0L,
 'rx_frame_errors': 0L,
 'rx_pkts': 0L,
 'rx_pkts_nic': 0,
 'rx_pps': 0L,
 'tx_bps': 0L,
 'tx_bytes': 6000L,
 'tx_bytes_nic': 0,
 'tx_pkts': 100L,
 'tx_pkts_nic': 0,
 'tx_pps': 0L}
rx stats:
{'rx_bps': 0L,
 'rx_bytes': 6000L,
 'rx_bytes_nic': 0,
 'rx_drops': 0L,
 'rx_errors': 0L,
 'rx_fifo_errors': 0L,
 'rx_frame_errors': 0L,
 'rx_pkts': 100L,
 'rx_pkts_nic': 0,
 'rx_pps': 0L,
 'tx_bps': 0L,
 'tx_bytes': 0L,
```

```
 'tx_bytes_nic': 0,
 'tx_pkts': 0L,
 'tx_pkts_nic': 0,
 'tx_pps': 0L}
saving capture as capture.pcap
```

CHAPTER 4

Miscellaneous

Working with other agents

Often, the drone instance to which we connect already has some ports and streams configured. We usually want to fetch this configuration. This is easy with the various `fetch` methods of the different objects:

- To fetch the ports (overriding `drone.ports`): `drone.fetch_ports()`
- To fetch the configuration of a specifig port (overriding any custom configuration the port object holds): `myport.fetch()`
- To fetch the streams configured on a port (overriding the streams in `myport.streams`): `myport.fetch_streams()`
- To fetch the configuration and the layers of a specifig stream (overriding any custom configuration/layers the stream object holds): `mystream.fetch()`

The symmetric operation, *i.e.* applying the configuration of the local objects on the remote drone instance, use the `save` methods:

- To apply a port configuration: `myport.save()`
- To apply a stream configuration: `mystream.save()`. Note that layers are a special case: when adding or deleting a layer, the operation is always applied on the remote drone instance.

Saving/Loading configurations

simple_ostinato.Port and *simple_ostinato.Stream* classes have a `to_dict()` and `from_dict()` method, which respectively dump and load the object configuration to/from a dictionary. It can be useful to save stream, or port configurations as json or yaml.

For example:

```
import json

# save the "my_port" configuration (including all the streams) in a file
with open('myport.json', 'w') as f:
    json.dump(myport.to_dict(), f)

# later, load this configuration for another port when loading a
# configuration, the `name` and `is_enable` keys are ignored, since they
# are readonly properties. this allows to re-use the same configuration on
# different ports.
with open('myport.json', 'r') as f:
    anotherport.from_dict(json.load(f))
anotherport.save()

# in case you only want to load the streams:
with open('myport.json', 'r') as f:
    anotherport.from_dict({'streams': json.load(f)['streams']})
anotherport.save()
```

CHAPTER 5

simple_ostinato package

class simple_ostinato.**Drone**(*host*, *connect=True*)
Bases: object

Wrapper for ostinato.core.DroneProxy.

All the protocol buffer related methods are prefixed with _o_ and are for internal use only.

Parameters

- **host** (*str*) – ip address or hostname of the host running the drone instance you want to connect to.
- **connect** (*bool*) – if True, attempt to connect to the remote instance when the object is initialized. Otherwise, it can be done manually later with *connect()*

connect()
Connect to the remote drone instance. By default, it is already called when the object is created.

disconnect()
Disconnect from the remote drone instance.

fetch_ports()
Get the list of all the ports on the remote host. They are stored in the ports dictionnary.

get_port(*name*)
Get ports from ports by name. If the port is not found, None is returned.

get_port_by_id(*port_id*)

reconnect()
Reconnect to the remote drone instance.

class simple_ostinato.**Port**(*drone*, *port_id*)
Bases: object

Represent a remote port. This class provides simple methods to add/remove streams, and send/capture traffic.

Parameters

- **drone** (*Drone*) – an object that wraps the underlying protocol buffer calls.

- **port_id** (*int*) – id of the port.

streams
dict – a dictionnary with all the streams configured on this port. It can be refreshed with *fetch_streams()*.

port_id
int – id of the port

add_stream(**layers*)
Create a new stream, on the remote drone instance, and return the corresponding Stream object. The object is also added to *streams*.

Layers must be instances of *simple_ostinato.protocols.Protocol*

```
>>> from simple_ostinato import protocols
>>> my_port.add_stream(protocols.Mac(), protocols.Ethernet())
```

clear_stats()
Clear the port statistics

del_stream(*stream_id*)
Delete the stream provided as argument.

Parameters **stream_id** (*int*) – id of the stream to delete from the port.

fetch()
Fetch the current port configuration from the remote drone instance.

fetch_streams()
Fetch the streams configured on this port, from the remote drone instance. The streams are stored in *streams*.

from_dict(*values*)

get_capture(*save_as=None*)
Get the lastest capture and return is as a string.

Parameters **save_as** (*str*) – if provided, the capture will also be saved as a pcap file at the specified location *on the host that runs drone*.

get_stats()
Fetch the port statistics, and return them as a dictionary.

get_stream(*stream_id*)
Return a the *Stream* object corresponding to the given stream ID (int)

get_streams_by_name(*name*)
Return a list of *Stream* s that have the given name (:class:str). Since most often names are unique, it is common to get a stream doing:

```
>>> my_stream_foo = my_port.get_streams_by_name('stream_foo')[0]
```

save()
Save the current port configuration on the remote drone instance.

save_capture(*o_capture_buffer*, *path*)

start_capture()
Start capturing. By default, this method is non-blocking and returns immediately, and *stop_send()* must be called to stop the capture.

start_send()
Start transmitting the streams that are enabled on this port.

stop_capture()
Stop the current capture

stop_send()
Stop sending

to_dict()

is_enabled
If `True` the port is enabled. Otherwise, it is disabled.

is_exclusive_control

name
Name of the port. This is a read-only attribute.

port_id
ID of the port. This is a read-only attribute.

transmit_mode
Can be `SEQUENTIAL` or `INTERLEAVED`.

user_name
Name of the port user.

class simple_ostinato.**Stream**(*port*, *stream_id*, *layers=None*)
Bases: object

Represent a stream configured on a port. Besides all the stream configuration parameters, a stream class has *layers* which define the packets to be sent.

Parameters

- **port** (*Port*) – the port instance on which the stream is defined.
- **stream_id** (*int*) – the stream ID.

disable()
Disable the stream. It is equivalent to setting *is_enabled* to `False`.

enable()
Enable the stream. It is equivalent to setting *is_enabled* to `True`.

fetch()
Fetch the stream configuration on the remote drone instance (including all the layers).

from_dict(*dictionary*)

save()
Save the current stream configuration (including the protocols).

to_dict()

bursts_per_sec
Number of bursts to send per second.

frame_len

frame_len_max

frame_len_min

is_enabled
Return `True` if the stream is enabled, `False` otherwise. By default, streams are not enabled.

layers
List of all the layers configured for this stream.

len_mode
Length mode. It must be either `FIXED` (the default), `INC`, `DEC` or `RANDOM`

mode
Sending mode. It must be either `FIXED` (the default) or `CONTINUOUS`.

If set to `FIXED`, a fixed number of packets or bursts is sent. If *unit* is set to `PACKETS`, then *num_packets* packets are sent. If it is set to `BURSTS` then *num_bursts* bursts are sent.

If set to `CONTINUOUS`, packets or bursts are sent continuously until the port stop transmitting.

name
Name of the stream (optional)

next
What to do after the current stream finishes. It is ignored if *mode* is set to `CONTINUOUS`.

- `STOP`: stop after this stream
- `GOTO_NEXT`: send the next enabled stream
- `GOTO_ID`: send a stream with a given ID.

num_bursts
Number of bursts to send. This is ignored if *mode* is set to `CONTINUOUS` or if *unit* is set to `PACKETS`.

num_packets
Number of packets to send. This is ignored if *mode* is set to `CONTINUOUS` or if *unit* is set to `BURSTS`.

packets_per_burst
Number of packets per burst. This is ignored if *mode* is set to `CONTINUOUS` or if *unit* is set to `PACKETS`

packets_per_sec
Number of bursts to send per second.

unit
Unit to send. It must be either `PACKETS` (the default) or `BURSTS`.

CHAPTER 6

simple_ostinato.protocols package

class simple_ostinato.protocols.**Protocol**(***kwargs*)
Base class for the actual protocols

class simple_ostinato.protocols.**Mac**(*source='00:00:00:00:00:00'*, *destination='FF:FF:FF:FF:FF:FF'*, ***kwargs*)
Represent the MAC layer. Since we make a distiction between the MAC layer and the Ethernet layer, this layer defines the source and destination MAC addresses.

from_dict(*dict_*)
Set the Mac layer configuration from a dictionary. Keys must be the same as the attributes names, and values but by valid values for these attributes.

to_dict()
Return the Mac layer configuration as a dictionnary.

destination
destination MAC address

destination_count
If *destination_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

destination_mode

destination_step
If *destination_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

source
Source MAC address

source_count
If *source_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

source_mode

source_step
If *source_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

class simple_ostinato.protocols.**Ethernet**(*ether_type='0x0800'*, ***kwargs*)
Represent the ethernet layer. Since we make a distinction between the MAC layer and the Ethernet layer, this layer only defines the ethernet type

from_dict(*dict_*)
Set the Ethernet layer configuration from a dictionary. Keys must be the same as the attributes names, and values but by valid values for these attributes.

to_dict()
Return the Ethernet layer configuration as a dictionnary.

ether_type
Ethernet type field. 0x800 is for IPv4 inner packets.. By default, this attribute is set automatically. Set *ether_type_override* to True to override this field

ether_type_count
If *ether_type_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

ether_type_mode
By default, *ether_type_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

ether_type_override

ether_type_step
If *ether_type_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

class simple_ostinato.protocols.**IPv4**(*flag_unused=0*, *dscp=0*, *flag_mf=0*, *ttl=127*, *protocol=0*, *header_length=5*, *fragments_offset=0*, *tos=0*, *destination='127.0.0.1'*, *source='127.0.0.1'*, *version=4*, *identification=0*, *checksum=0*, *flag_df=0*, *total_length=0*, ***kwargs*)
Represent the IPv4 layer.

from_dict(*dict_*)
Set the IPv4 layer configuration from a dictionary. Keys must be the same as the attributes names, and values but by valid values for these attributes.

to_dict()
Return the IPv4 layer configuration as a dictionnary.

checksum
Header checksum. By default, this attribute is set automatically. Set *checksum_override* to True to override this field

checksum_count
If *checksum_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

checksum_mode
By default, *checksum_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

checksum_override

checksum_step
If *checksum_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

destination

destination_count
If *destination_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

destination_mode
By default, *destination_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

destination_step
If *destination_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

dscp
Differentiated Services Code Point (DSCP) field (previously known as Type Of Service (TOS) field

dscp_count
If *dscp_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

dscp_mode
By default, *dscp_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

dscp_step
If *dscp_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_df
The "Don't Fragment" (DF) 1 bit flag

flag_df_count
If *flag_df_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_df_mode
By default, *flag_df_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_df_step
If *flag_df_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_mf
The "More Fragments" (MF) 1 bit flag

flag_mf_count
If *flag_mf_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_mf_mode
By default, *flag_mf_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_mf_step
If *flag_mf_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_unused
A 1 bit unused flag

flag_unused_count
If *flag_unused_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_unused_mode
By default, *flag_unused_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_unused_step
If *flag_unused_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

fragments_offset
The Fragment Offset field indicates the offset of a packet fragment in the original IP packet

fragments_offset_count
If *fragments_offset_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

fragments_offset_mode
By default, *fragments_offset_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

fragments_offset_step
If *fragments_offset_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

header_length
Internet Header Length (IHL) – number of 4 bytes words in the header. The minimum valid value is 5, and maximum valid value is 15.. By default, this attribute is set automatically. Set *header_length_override* to True to override this field

header_length_count
If *header_length_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

header_length_mode
By default, *header_length_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

header_length_override

header_length_step
If *header_length_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

identification
Identification field. This is used to identify packet fragments

identification_count
If *identification_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

identification_mode
By default, *identification_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

identification_step
If *identification_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

protocol
Indicates the protocol that is encapsulated in the IP packet.. By default, this attribute is set automatically. Set *protocol_override* to True to override this field

protocol_count
If *protocol_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

protocol_mode
By default, *protocol_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

protocol_override

protocol_step
If *protocol_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

source

source_count
If *source_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

source_mode
By default, *source_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

source_step
If *source_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

tos
Type Of Service (TOS) field. This field is now the Differentiated Services Code Point (DSCP) field.

tos_count
If *tos_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

tos_mode
By default, *tos_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

tos_step
If *tos_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

total_length
Total length of the IP packet in bytes. The minimum valid value is 20, and the maxium is 65,535. By default, this attribute is set automatically. Set *total_length_override* to True to override this field

total_length_count
If *total_length_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

total_length_mode
By default, *total_length_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

total_length_override

total_length_step
If *total_length_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

ttl
Time To Live (TTL) field.

ttl_count
If *ttl_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

ttl_mode
By default, *ttl_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

ttl_step
If *ttl_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

version
Version of the protocol (usually 4 or 6). By default, this attribute is set automatically. Set *version_override* to True to override this field

version_count
If *version_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

version_mode
By default, *version_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

version_override

version_step
If *version_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

class simple_ostinato.protocols.**Payload**(*pattern='00 00 00 00'*, *mode='FIXED_WORD'*, ***kwargs*)

Base class for the actual protocols

from_dict(*values*)

to_dict()

mode
The mode can be one of –

- DECREMENT_BYTE
- FIXED_WORD
- INCREMENT_BYTE
- RANDOM

pattern
Payload initial word. Depending on the chosen mode, this word will be repeated unchanged, incremented/decremented, or randomized

class simple_ostinato.protocols.**Tcp**(*flag_ack=0*, *header_length=0*, *reserved=0*, *ack_num=0*, *flag_rst=0*, *window_size=0*, *destination=49153*, *flag_psh=0*, *urgent_pointer=0*, *source=49152*, *flag_ece=0*, *flag_urg=0*, *sequence_num=0*, *checksum=0*, *flag_syn=0*, *flag_cwr=0*, *flag_fin=0*, *flag_ns=0*, ***kwargs*)

Represent an TCP datagram

from_dict(*dict_*)
Set the Tcp layer configuration from a dictionary. Keys must be the same as the attributes names, and values but by valid values for these attributes.

to_dict()
Return the Tcp layer configuration as a dictionnary.

ack_num
Acknowledgement number

ack_num_count
If *ack_num_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

ack_num_mode
By default, *ack_num_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

ack_num_step
If *ack_num_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

checksum
Checksum of the datagram, calculated based on the IP pseudo-header. Its meaning depends on the value og the ack flag.. By default, this attribute is set automatically. Set *checksum_override* to True to override this field

checksum_count
If *checksum_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

checksum_mode
By default, *checksum_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

checksum_override

checksum_step
If *checksum_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

destination
Destination port number. By default, this attribute is set automatically. Set *destination_override* to True to override this field

destination_count
If *destination_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

destination_mode
By default, *destination_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

destination_override

destination_step
If *destination_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_ack
ACK flag

flag_ack_count
If *flag_ack_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_ack_mode
By default, *flag_ack_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_ack_step
If *flag_ack_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_cwr
Congestion Window Reduced flag

flag_cwr_count
If *flag_cwr_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_cwr_mode
By default, *flag_cwr_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_cwr_step
If *flag_cwr_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_ece
ECN-Echo flag. Its meaning depends on the syn field value.

flag_ece_count
If *flag_ece_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_ece_mode
By default, *flag_ece_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_ece_step
If *flag_ece_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_fin
No more data from sender

flag_fin_count
If *flag_fin_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_fin_mode
By default, *flag_fin_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_fin_step
If *flag_fin_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_ns
ECN-nonce concealment protection (experimental). By default, this attribute is set automatically. Set *flag_ns_override* to True to override this field

flag_ns_count
If *flag_ns_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_ns_mode
By default, *flag_ns_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_ns_override

flag_ns_step
If *flag_ns_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_psh
Push function

flag_psh_count
If *flag_psh_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_psh_mode
By default, *flag_psh_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_psh_step
If *flag_psh_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_rst
Reset the connection

flag_rst_count
If *flag_rst_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_rst_mode
By default, *flag_rst_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_rst_step
If *flag_rst_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_syn
Synchronize sequence numbers

flag_syn_count
If *flag_syn_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_syn_mode
By default, *flag_syn_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_syn_step
If *flag_syn_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

flag_urg
Urgent pointer flag.

flag_urg_count
If *flag_urg_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

flag_urg_mode
By default, *flag_urg_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

flag_urg_step
If *flag_urg_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

header_length
Size of the TCP header in 4 bytes words. This field is also known as "Data offset". By default, this attribute is set automatically. Set *header_length_override* to True to override this field

header_length_count
If *header_length_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

header_length_mode
By default, *header_length_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

header_length_override

header_length_step
If *header_length_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

reserved
Reserved for future use and must be set to 0. By default, this attribute is set automatically. Set *reserved_override* to True to override this field

reserved_count
If *reserved_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

reserved_mode
By default, *reserved_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

reserved_override

reserved_step
If *reserved_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

sequence_num
Sequence number of the datagram. Its meaning depends on the syn flag value.

sequence_num_count
If *sequence_num_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

sequence_num_mode
By default, *sequence_num_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

sequence_num_step
If *sequence_num_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

source
Source port number. By default, this attribute is set automatically. Set *source_override* to True to override this field

source_count
If *source_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

source_mode
By default, *source_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

source_override

source_step
If *source_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

urgent_pointer
Urgent pointer.

urgent_pointer_count
If *urgent_pointer_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

urgent_pointer_mode
By default, *urgent_pointer_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

urgent_pointer_step
If *urgent_pointer_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

window_size
Size of the receive window, which specifies the number of window size units that the sender of this segment is currently willing to receive

window_size_count
If *window_size_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

window_size_mode
By default, *window_size_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

window_size_step
If *window_size_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

class simple_ostinato.protocols.**Udp**(*source=49152*, *length=0*, *destination=49153*, *checksum=0*, ***kwargs*)
Represent an UDP datagram

from_dict(*dict_*)
Set the Udp layer configuration from a dictionary. Keys must be the same as the attributes names, and values but by valid values for these attributes.

to_dict()
Return the Udp layer configuration as a dictionnary.

checksum
Checksum of the datagram, calculated based on the IP pseudo-header.. By default, this attribute is set automatically. Set *checksum_override* to True to override this field

checksum_count
If *checksum_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

checksum_mode
By default, *checksum_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

checksum_override

checksum_step
If *checksum_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

destination
Destination port number. By default, this attribute is set automatically. Set *destination_override* to True to override this field

destination_count
If *destination_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

destination_mode
By default, *destination_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

destination_override

destination_step
If *destination_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

length
Length of the UDP datagram (header and payload).. By default, this attribute is set automatically. Set *length_override* to True to override this field

length_count
If *length_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

length_mode
By default, *length_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

length_override

length_step
If *length_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

source
Source port number. By default, this attribute is set automatically. Set *source_override* to True to override this field

source_count
If *source_mode* is INCREMENT, DECREMENT, specifies the number of packets before resetting the field to its initial value.

source_mode
By default, *source_mode* is FIXED. Possible values are: INCREMENT, DECREMENT, RANDOM, FIXED.

source_override

source_step
If *source_mode* is set to INCREMENT or DECREMENT, specifies the increment or decrement step.

Python Module Index

S

Index

E

F

G

H

I

L

M

N

P

R

S

T

U

V

W