```
import struct
import socket
import random
from time import sleep, time
from dataclasses import dataclass, field
from collections import Counter
import pytest
from new_monitor import *


class DummyWifiDevicesMonitor(WifiDevicesMonitor):
    def __init__(self, cache_duration=10 * 60):
        self._statistics = Statistics()
        self._devices = {}
        self._cache_duration = cache_duration
        self._last_cache_cleanup = time()


def dummy_clean_cache():
    """Dummy function for unit testing"""
    for mac in mon._devices:
        if mon._devices[mac] > time() + mon._cache_duration:
            del mon._devices[mac]


def dummy_format_mac(bytes_addr):
    """Dummy function for unit testing"""
    bytes_s = map('{:02x}'.format, bytes_addr)
    return "".join(bytes_s).lower()


def dummy_add_mac(mac):
    """Dummy function for unit testing"""
    if mac == "ffffffffffff":
        return

    if mac == '000000000000':
        return

    mon._devices[mac] = time()
```

```
        def dummy_handle_packet(raw_packet):
            """Dummy function for unit testing"""
            try:
                dest_mac, src_mac, bssid, proto = struct.unpack('! 6s 6s 6s H', raw_packet[22:42])
                for mac in dest_mac, src_mac, bssid:
                    # print(mac)
                    mac = dummy_format_mac(mac)
                    dummy_add_mac(mac)

            except Exception as e:
                pass  # Ignore packets without addresses
            dummy_clean_cache()


        mon = DummyWifiDevicesMonitor()  # Initialization
        for mac in ("0123456789ab", "aaaaaaaaaaaa", "abcdefffffff", "fdfdfafvfcff"):
            mon._devices[mac] = random.randrange(1600771000, 1600771144)


        def test_devices_in_timespan():
            for timespan in (0, 0.5, 3.6, 50, 100):
                devices_in_timespan = []
                for mac in mon._devices:
                    if mon._devices[mac] > time() - timespan:
                        devices_in_timespan.append(mac)
                assert devices_in_timespan == mon._devices_in_timespan(timespan)


        def test_clean_cache():
            mon._clean_cache()
            for mac in mon._devices:
                assert round(mon._devices[mac], 2) <= round(time(), 2) + mon._cache_duration


        def test_format_mac():
            for address in (
                    b'0\x06\xe8\xb5\xb9\x1d', b'\xd7\x80\x07l\x01Z', b'\x00\x01\xef\xff\xff\xfa',
b'\x04\x11\xc4\xe5\xc0\xa8'):
```

```
            assert mon._format_mac(address) == "".join(map('{:02x}'.format, address)).lower()


        def test_add_mac():
            for mac in ("0123456789ab", "aaaaaaaaaaaa", "abcdefffffff", "fdfdfafvfcff"):
                mon._add_mac(mac)
                assert round(mon._devices[mac], 2) == round(time(), 2)
            for mac in ('000000000000', "ffffffffffff"):
                mon._add_mac(mac)
                try:
                    assert mon._devices[mac] is None
                except KeyError:
                    pass


        def test_handle_packet():
            for raw_packet in DummyData:
                mon._devices = {}
                intended_devices = {}
                try:
                    dest_mac, src_mac, bssid, proto = struct.unpack('! 6s 6s 6s H',
raw_packet[22:42])
                    for mac in dest_mac, src_mac, bssid:
                        # print(mac)
                        mac = dummy._format_mac(mac)
                        dummy._add_mac(mac)

                except Exception:
                    pass
                intended_devices = mon._devices
                mon._devices = {}

                mon._handle_packet(raw_packet)
                for key in intended_devices:
                    intended_devices[key] = round(intended_devices[key], 2) + 1
                for key in mon._devices:
                    mon._devices[key] = round(mon._devices[key], 2)
                assert intended_devices == mon._devices
```

```
        def test_parse_raw_socket_output():
            print(mon._devices, mon._statistics)
            start = datetime.datetime.utcnow()
            dummy_index = 0
            while datetime.datetime.utcnow() < start + datetime.timedelta(seconds=TIMEOUT):
                try:
                    raw_data, addr = DummyData[dummy_index]
                    dummy_index += 1
                except IndexError:
                    pass
                mon._statistics.packets_read += 1
                if len(raw_data) < 48:
                    continue
                dummy_handle_packet(raw_data)
            intended_devices = mon._devices
            intended_packets_read = mon._statistics.packets_read
            mon._statistics = Statistics()
            mon._devices = {}  # Saves the correct values and resets

            mon._parse_raw_socket_output(True, DummyData)
            for key in intended_devices:
                intended_devices[key] = round(intended_devices[key], 2) + 1
            for key in mon._devices:
                mon._devices[key] = round(mon._devices[key], 2)
            assert intended_devices == mon._devices
            assert round(intended_packets_read/10000) ==
round(mon._statistics.packets_read/10000)
            """
             Compares tens of thousands.
             Left having a couple thousand more packets is normal, because dummy function is
somehow faster
             """



        def test_get_devices_for_duration():
            for timespan in (0, 0.5, 3.6, 50, 100):
                assert len(mon._devices_in_timespan(timespan)) ==
mon.get_devices_for_duration(timespan)
```

```
        def test_get_devices_by_vendor_for_duration():
            for timespan in (0, 0.5, 3.6, 50, 100):
                all_devices = mon._devices_in_timespan(timespan)
                vendors = []
                for device in all_devices:
                    vendors.append(device[:6])
                intended_result = dict(Counter(vendors))
                assert mon.get_devices_by_vendor_for_duration(timespan) == intended_result


        def test_get_devices():
            assert mon.get_devices() == list(mon._devices.keys())


        def test_get_statistics():
            assert mon.get_statistics() == mon._statistics


        def integration_test():
            """
            Integrated test for:

_parse_raw_socket_output(),_handle_packet(),_add_mac(),_format_mac(),_clean_cache()
            """
            start = datetime.datetime.utcnow()
            dummy_index = 0
            while datetime.datetime.utcnow() < start + datetime.timedelta(seconds=TIMEOUT):
                try:
                    raw_data, addr = DummyData[dummy_index]
                    dummy_index += 1
                except IndexError:
                    pass
                mon._statistics.packets_read += 1
                if len(raw_data) < 48:
                    continue
                mon._handle_packet(raw_data)
            intended_devices = mon._devices
            intended_packets_read = mon._statistics.packets_read
            mon._statistics = Statistics()
            mon._devices = {}  # Saves the correct values and resets
```

```
            mon._parse_raw_socket_output(True, DummyData)
            for key in intended_devices:
                intended_devices[key] = round(intended_devices[key], 2)
            for key in mon._devices:
                mon._devices[key] = round(mon._devices[key], 2)
            assert intended_devices == mon._devices
            assert intended_packets_read == mon._statistics.packets_read


        DummyData = (
            (

b'\x01\x00^\x7f\xff\xfa\xec\x08kkX`\x08\x00E\x00\x01$\x00\x00@\x00\x04\x11\xc5%\xc0\xa8\x
00\x01\xef\xff\xff\xfa\xd7\x80\x07l\x01\x10\x81\x86NOTIFY * HTTP/1.1\r\nHOST:
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=100\r\nLOCATION:
http://192.168.0.1:1900/igd.xml\r\nNT: upnp:rootdevice\r\nNTS: ssdp:alive\r\nSERVER:
ipos/7.0 UPnP/1.0 TL-WR740N/5.0\r\nUSN:
uuid:060b7353-fca6-4070-85f4-1fbfb9add62c::upnp:rootdevice\r\n\r\n',
                ('wlxc025e918a6ba', 2048, 2, 1, b'\xec\x08kkX`')),

            (

b'\x01\x00^\x7f\xff\xfa\xec\x08kkX`\x08\x00E\x00\x01-\x00\x00@\x00\x04\x11\xc5\x1c\xc0\xa8\
x00\x01\xef\xff\xff\xfa\xd7\x80\x07l\x01\x19\xbc\x81NOTIFY * HTTP/1.1\r\nHOST:
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=100\r\nLOCATION:
http://192.168.0.1:1900/igd.xml\r\nNT: uuid:060b7353-fca6-4070-85f4-1fbfb9add62c\r\nNTS:
ssdp:alive\r\nSERVER: ipos/7.0 UPnP/1.0 TL-WR740N/5.0\r\nUSN:
uuid:060b7353-fca6-4070-85f4-1fbfb9add62c\r\n\r\n',
                ('wlxc025e918a6ba', 2048, 2, 1, b'\xec\x08kkX`')),

            (

b'\x01\x00^\x7f\xff\xfa\xec\x08kkX`\x08\x00E\x00\x01l\x00\x00@\x00\x04\x11\xc4\xdd\xc0\xa8\
x00\x01\xef\xff\xff\xfa\xd7\x80\x07l\x01X\x12\x88NOTIFY * HTTP/1.1\r\nHOST:
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=100\r\nLOCATION:
http://192.168.0.1:1900/igd.xml\r\nNT:
urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\nNTS: ssdp:alive\r\nSERVER:
ipos/7.0 UPnP/1.0 TL-WR740N/5.0\r\nUSN:
```

uuid:060b7353-fca6-4070-85f4-1fbfb9add62c::urn:schemas-upnp-org:device:InternetGateway
Device:1\r\n\r\n',
('wlxc025e918a6ba', 2048, 2, 1, b'\xec\x08kkX`')),

(

b'\x01\x00^\x7f\xff\xfa\xec\x08kkX`\x08\x00E\x00\x01d\x00\x00@\x00\x04\x11\xc4\xe5\xc0\xa8
\x00\x01\xef\xff\xff\xfa\xd7\x80\x07l\x01P\xbb@NOTIFY * HTTP/1.1\r\nHOST:
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=100\r\nLOCATION:
http://192.168.0.1:1900/igd.xml\r\nNT:
urn:schemas-upnp-org:service:Layer3Forwarding:1\r\nNTS: ssdp:alive\r\nSERVER: ipos/7.0
UPnP/1.0 TL-WR740N/5.0\r\nUSN:
uuid:060b7353-fca6-4070-85f4-1fbfb9add62c::urn:schemas-upnp-org:service:Layer3Forwardin
g:1\r\n\r\n',
('wlxc025e918a6ba', 2048, 2, 1, b'\xec\x08kkX`')),

(

b'\x01\x00^\x7f\xff\xfa\xec\x08kkX`\x08\x00E\x00\x01-\x00\x00@\x00\x04\x11\xc5\x1c\xc0\xa8\
x00\x01\xef\xff\xff\xfa\xd7\x80\x07l\x01\x19nPNOTIFY * HTTP/1.1\r\nHOST:
239.255.255.250:1900\r\nCACHE-CONTROL: max-age=100\r\nLOCATION:
http://192.168.0.1:1900/igd.xml\r\nNT: uuid:254e9977-8964-49f3-b8d5-51acb7bd40fc\r\nNTS:
ssdp:alive\r\nSERVER: ipos/7.0 UPnP/1.0 TL-WR740N/5.0\r\nUSN:
uuid:254e9977-8964-49f3-b8d5-51acb7bd40fc\r\n\r\n',
('wlxc025e918a6ba', 2048, 2, 1, b'\xec\x08kkX`'))
)