EEE413-DATA COMMUNICATION-LAB2

Author: Zhipeng Ye

Student ID: 1926908

Faculty: MSc Multimedia Telecommunications

Date: 18 December 2019

Catalog

Introduction

In the class, we have studied leaky bucket and token bucket theory which is a theory about controling flux passing through the system and promoting the quality of service.

Leaky bucket controls the burst size (Figure 1). This means the packet will be cut down when burst size is larger than the size of leaky bucket.

Moreover, token bucket controls the peak rate {Figure 2}. When the token bucket has enough token, the packet can pass through system.

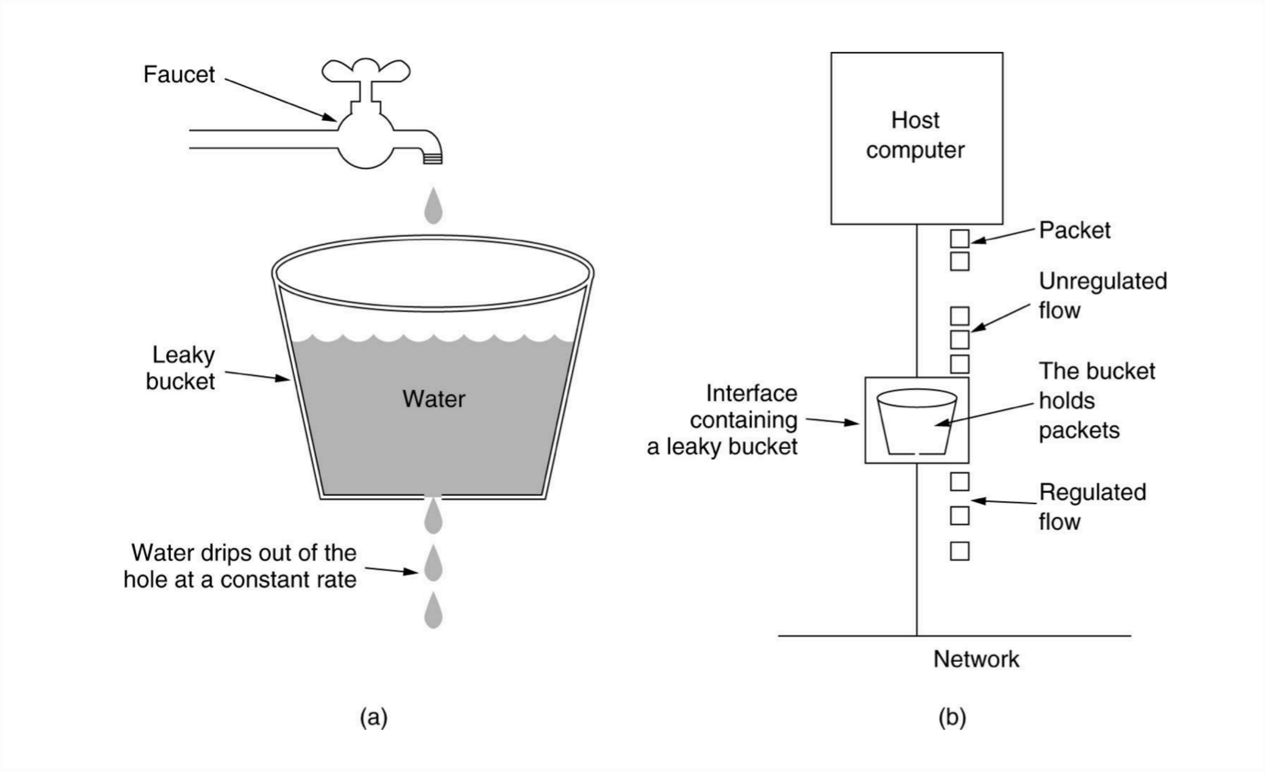


Figure 1

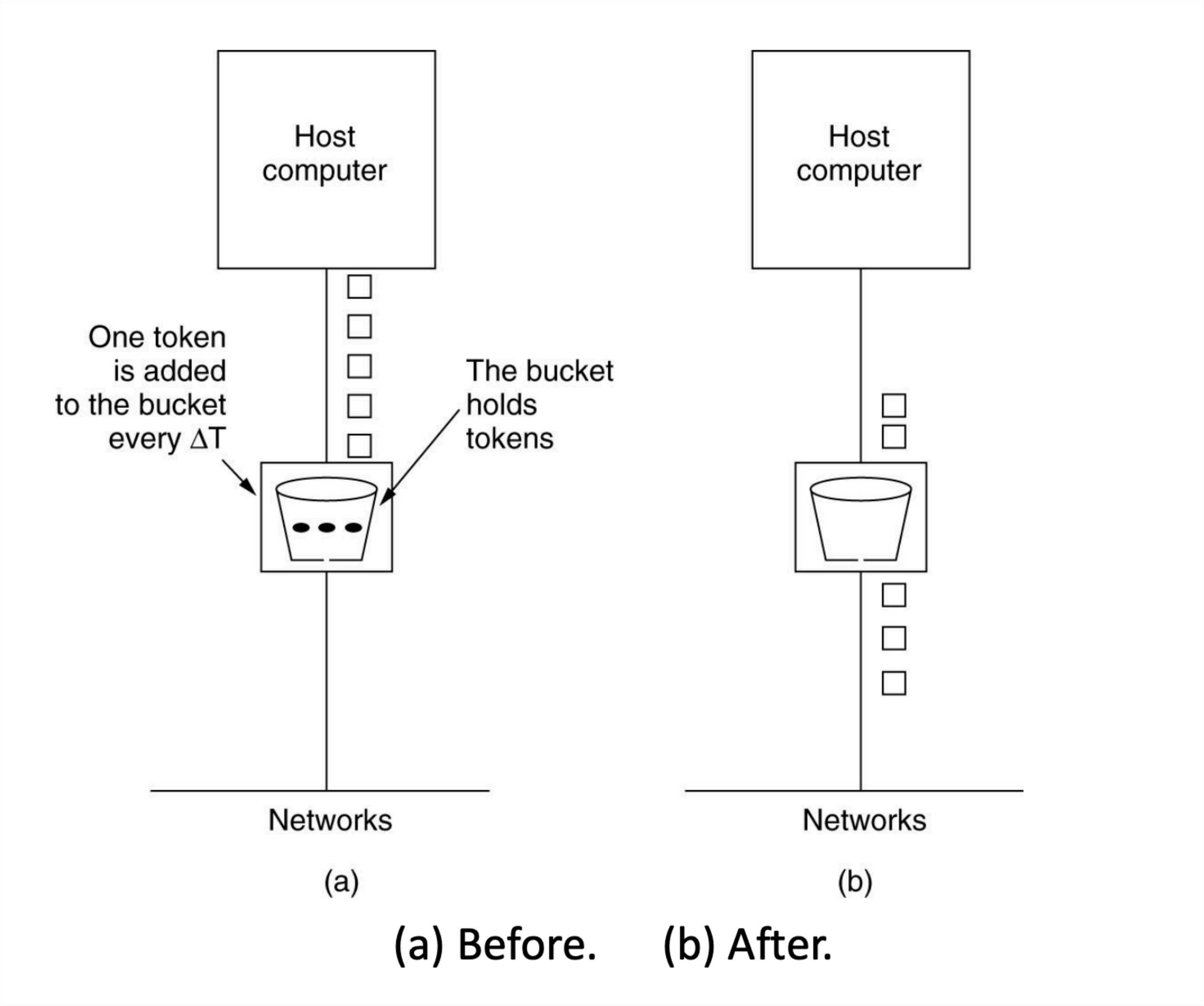


Figure 2

Calculation

In this assignment, we focus on the Token bucket filter (TBF) (Figure 3).

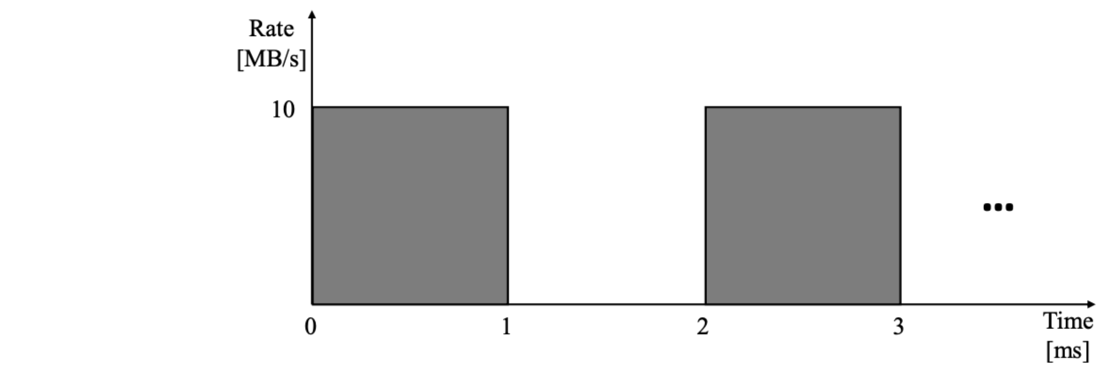


Fig. 1. A periodic on-off traffic pattern.

Figure 3

Here are the Token Bucket Filter parameters:

- Token generation rate: ρ MB/s.
- Token bucket capacity: C MB (full when the first burst arrives)

- Peak output rate: 10 MB/s (same as the input rate)

For designing the parameters of Token bucket Filter, we must do threes steps to do experiments correctly, using a fluid model (In a fluid model, traffic arrivals are assumed to be continuous like liquid flows rather than discrete).

First of all, as we discussed in the lecture, the burst length S is given by $S = \frac{C}{M-\rho}$.

Because of $\rho = 10MB/s$, $S = \frac{C}{10-\rho}$.

As shown in Figure 3, the duration of burst is 0.001s (i.e., 1ms). we must make sure the first burst can pass through without no shaping.

$$S = \frac{C}{10-\rho} \geq 0.001$$
$$C \geq 0.01 - 0.001\rho$$

So we get the minimum token bucket capacity $0.01 - 0.001\rho$ MB.

Secondly, for avoiding burst generation rate is larger than token bucket generation rate. We must make the amount tokens generated during the off period should be equal to or greater than the token bucket capacity.

$$0.001\rho \geq 0.01 - 0.001\rho$$
$$\rho \geq 5$$

Therefore, we get the minimum token generation rate is $5MB/s$.

Finally, we obtain the minimum token bucket capacity $5kB$.

Simulation

In this simulation, we use simpy to simulate token bucket filter.

Here is my python code (change the file supported by teacher), I convert uint from MB/s into b/ms for conveniency.

Integrated Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     queue_onoff_traffic.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2018-11-23
#
# @brief    Simulate a queueing system with an on-off packet generator.
#


import argparse
import numpy as np
import simpy


class Packet(object):
    """
    Parameters:
    - ctime: packet creation time
    - size: packet size in bytes
    """

    def __init__(self, ctime, size):
        self.ctime = ctime
        self.size = size


class OnoffPacketGenerator(object):
    """Generate fixed-size packets back to back based on on-off status.

    Parameters:
    - env: simpy.Environment
    - pkt_size: packet size in bytes
    - pkt_ia_time: packet interarrival time in second
    - on_period: ON period in second
    - off_period: OFF period in second
    """

    def __init__(self, env, pkt_size, pkt_ia_time, on_period, off_period,
                 trace=False):
        self.env = env
        self.pkt_size = pkt_size
        self.pkt_ia_time = pkt_ia_time
        self.on_period = on_period
        self.off_period = off_period
        self.trace = trace
        self.out = None
        self.on = True
        self.gen_permission = simpy.Resource(env, capacity=1)
        self.action = env.process(self.run())  # start the run process when
an instance is created

    def run(self):
        env.process(self.update_status())
        while True:
            with self.gen_permission.request() as req:
                yield req
                p = Packet(self.env.now, self.pkt_size)
                self.out.put(p)
                if self.trace:
                    print("t={0:.4E} [s]: packet generated with size={1:.4E}
[B]".format(self.env.now, self.pkt_size))
            yield self.env.timeout(self.pkt_ia_time)

    def update_status(self):
        while True:
            now = self.env.now
            if self.on:
                if self.trace:
                    print("t={:.4E} [s]: OFF->ON".format(now))
                yield env.timeout(self.on_period)
            else:
                if self.trace:
                    print("t={:.4E} [s]: ON->OFF".format(now))
                req = self.gen_permission.request()
                yield env.timeout(self.off_period)
                self.gen_permission.release(req)
            self.on = not self.on  # toggle the status


class FifoQueue(object):
    """Receive, process, and send out packets.

    Parameters:
    - env : simpy.Environment
    """

    def __init__(self, env, trace=False):
        self.trace = trace
        self.store = simpy.Store(env)
        self.env = env
        self.out = None
        self.action = env.process(self.run())

        # unit b/ms
        self.transmission_rate = 10 * 10 ** 3
        self.token_rate = 5 * 10 ** 3
        self.capacity = 5 * 10 ** 3
        self.token_amount = self.capacity
        self.current_time = env.now

    def run(self):

        while True:
            msg = (yield self.store.get())

            now = env.now
            time_passed = now - self.current_time

            self.token_amount = self.token_amount + self.token_rate *
time_passed

            if self.token_amount > self.capacity:
                self.token_amount = self.capacity

            if msg.size > self.token_amount:

                token_difference = msg.size - self.token_amount
                yield self.env.timeout(token_difference / self.token_rate)
                self.token_amount = 0
            else:
                self.token_amount = self.token_amount - msg.size

            self.current_time = env.now

            # because of the blocking model, teacher told us remove
transmission time
            # delay_time = msg.size / self.transmission_rate
            # yield self.env.timeout(delay_time)

            self.out.put(msg)

    def put(self, pkt):
        self.store.put(pkt)


class PacketSink(object):
    """Receives packets and display delay information.

    Parameters:
    - env : simpy.Environment
    - trace: Boolean

    """

    def __init__(self, env, trace=False):
        self.store = simpy.Store(env)
        self.env = env
        self.trace = trace
        self.wait_times = []
        self.action = env.process(self.run())

    def run(self):
        while True:
            msg = (yield self.store.get())
            now = self.env.now
            msg.delay = now - msg.ctime

            # append packet object into list[]
            self.wait_times.append(msg)
            if self.trace:
                print("t={0:.4E} [s]: packet arrived with size={1:.4E}
[B]".format(now, msg.size))

    def put(self, pkt):
        self.store.put(pkt)


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-S",
        "--pkt_size",
        help="packet size [byte]; default is 100",
        default=100,
        type=int)
    parser.add_argument(
        "-A",
        "--pkt_ia_time",
        help="packet interarrival time [second]; default is 10 * 10 ** (-3)",
        default=10 * 10 ** (-3),
        type=float)
    parser.add_argument(
        "--on_period",
        help="on period [m second]; default is 1.0",
        default=1,
        type=float)
    parser.add_argument(
        "--off_period",
        help="off period [m second]; default is 1.0",
        default=1,
        type=float)
    parser.add_argument(
        "-T",
        "--sim_time",
        help="time to end the simulation [m second]; default is 2",
        default=2,
        type=float)
    parser.add_argument(
        "-R",
        "--random_seed",
        help="seed for random number generation; default is 1234",
        default=1234,
        type=int)
    parser.add_argument('--trace', dest='trace', action='store_true')
    parser.add_argument('--no-trace', dest='trace', action='store_false')
    parser.set_defaults(trace=True)
    args = parser.parse_args()

    # set variables using command-line arguments
    pkt_size = args.pkt_size
    pkt_ia_time = args.pkt_ia_time
    on_period = args.on_period
    off_period = args.off_period
    sim_time = args.sim_time
    random_seed = args.random_seed
    trace = args.trace

    env = simpy.Environment()
    pg = OnoffPacketGenerator(env, pkt_size, pkt_ia_time, on_period,
off_period,
                              trace)
    fifo = FifoQueue(env, trace)
    ps = PacketSink(env, trace)
    pg.out = fifo
    fifo.out = ps
    env.run(until=sim_time)

    print('Each packet delay time:')
    # sorting by ctime
    ps.wait_times = sorted(ps.wait_times, key=lambda Packet: Packet.ctime)

    # calculate difference by each packet, because of blocking model
    list_msg = []
    for i in range(len(ps.wait_times) - 1):
        msg_after = ps.wait_times[i + 1]
        msg_before = ps.wait_times[i]

        if i == 0:
            difference = msg_before.ctime - 0
        else:
            difference = msg_after.delay - msg_before.delay

        # because of on-off model
        if difference < 0:
            difference = 0

        packet = Packet(msg_before.ctime, msg_before.size)
        packet.delay = difference

        list_msg.append(packet)

    for msg in list_msg:
        # print("msg ctime : {:.4E}, msg delay : {:.4E}".format(msg.ctime,
msg.delay))
        print("{:.4E},{:.4E}".format(msg.ctime, msg.delay))

    print("Average waiting time:")
    print("{:.4E} = {:.4E}\n".format(pkt_size, np.mean([msg.delay for msg in
list_msg])))
```

Plot Code

```
#!/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: plot_each_delay.py
# @time: 2019-12-19 23:00
# @desc:

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.pyplot import MultipleLocator

if __name__ == "__main__":
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)

    x_major_locator = MultipleLocator(1)

    ax.xaxis.set_major_locator(x_major_locator)
    # Or if you want different settings for the grids:
    ax.grid(which='minor', alpha=0.2)
    ax.grid(which='major', alpha=0.5)

    # calculate and plot analytical results

    # load and plot simulation results
    # change delimiter '/t' into ','
    x, y = np.loadtxt('eachpacket_waitingtime_data200.out', delimiter=',',
unpack=True)

    data_frame = {}

    for i in range(len(x)):
        data_frame[x[i]] = y[i]

    xlab = np.arange(min(x), max(x) + 0.01, 0.01).tolist()

    ylab = []

    for x in xlab:

        y_value = data_frame.get(round(x,5))
        if y_value == None:
            ylab.append(0)
        else:
            ylab.append(y_value)

    plt.plot(xlab, ylab)
    plt.xlabel('each packet generation timestamp ms')
    plt.ylabel('waiting time ms')

    # add labels, legend, and title
    plt.legend()
    plt.title(r'each packet waiting time')

    plt.savefig('each packet waiting time.pdf')
    plt.show()
```

Explanation

The main change implementing token bucket filter is

```
    def __init__(self, env, trace=False):
        self.trace = trace
        self.store = simpy.Store(env)
        self.env = env
        self.out = None
        self.action = env.process(self.run())

        # unit b/ms
        self.transmission_rate = 10 * 10 ** 3
        self.token_rate = 5 * 10 ** 3
        self.capacity = 5 * 10 ** 3
        self.token_amount = self.capacity
        self.current_time = env.now

    def run(self):

        while True:
            msg = (yield self.store.get())

            now = env.now
            time_passed = now - self.current_time

            self.token_amount = self.token_amount + self.token_rate * 
time_passed

            if self.token_amount > self.capacity:
                self.token_amount = self.capacity

            if msg.size > self.token_amount:

                token_difference = msg.size - self.token_amount
                yield self.env.timeout(token_difference / self.token_rate)
                self.token_amount = 0
            else:
                self.token_amount = self.token_amount - msg.size

            self.current_time = env.now

            # because of the blocking model, teacher told us remove 
transmission time
            # delay_time = msg.size / self.transmission_rate
            # yield self.env.timeout(delay_time)

            self.out.put(msg)

    def put(self, pkt):
        self.store.put(pkt)
```

I use the simpy.store to implement FIFO queue. According to the parameters I have calculated, I design a token bucket filter.

Firstly, I compute the amount of token by passed time and token generation rate and if token amount larger than the capacity, the buffer will discard some token.

Secondly, if packet.size is larger than the amount of token, the packet will wait for token generation. If the pakcet size is samller than the amount of token, the packet don't need to wait for generating token.

Finally, I add the time costing in transmission. However, It is a blocking model. As mentioned in class, teacher stress that we should remove the transmission time. Therefore, I remove this part.

Other changes in my code

```
    def run(self):
        while True:
            msg = (yield self.store.get())
            now = self.env.now
            msg.delay = now - msg.ctime

            # append packet object into list[]
            self.wait_times.append(msg)
            if self.trace:
                print("t={0:.4E} [s]: packet arrived with size={1:.4E}
[B]".format(now, msg.size))
```

add packet into list

```
    print('Each packet delay time:')

    # sorting by ctime
    ps.wait_times = sorted(ps.wait_times, key=lambda Packet: Packet.ctime)

    # calculate difference by each packet, because of blocking model
    list_msg = []
    for i in range(len(ps.wait_times) - 1):
        msg_after = ps.wait_times[i + 1]
        msg_before = ps.wait_times[i]

        if i == 0:
            difference = msg_before.ctime - 0
        else:
            difference = msg_after.delay - msg_before.delay

        # because of on-off model
        if difference < 0:
            difference = 0

        packet = Packet(msg_before.ctime, msg_before.size)
        packet.delay = difference

        list_msg.append(packet)

    for msg in list_msg:
        # print("msg ctime : {:.4E}, msg delay : {:.4E}".format(msg.ctime,
msg.delay))
        print("{:.4E},{:.4E}".format(msg.ctime, msg.delay))

    print("Average waiting time:")
    # print("{:.4E} = {:.4E}\n".format(pkt_size, np.mean([msg.delay for msg
in list_msg])))
    print("packet size:{:.4E} , average delay:{:.4E}\n".format(pkt_size,
np.mean([msg.delay for msg in list_msg])))
```

calculate each packet waiting time

Explaination

The reason for changing this part is that I want to compare each packet waiting time. Therefore, I add a attribute - delay in packet object to record waiting time. Moreover, I find the experiment exists system error, because the latter packet should wait until previous packets have arrived due to the blocking I/O model. The process can be decribed by this formula.

$$\begin{aligned} f(packet_1) &= token\_delay_1 \\ f(packet_2) &= f(packet_1) + token\_delay_2 \\ f(packet_3) &= f(packet_2) + token\_delay_3 \\ &\cdots \end{aligned}$$

Therefore, If I want to know the each pakcet delay time. I can use subtract to remove the effect from previous packet.

Finally, I print the each packet delay.

The main algorithm has been completed. Therefore, I can use command line to run it.

Execute

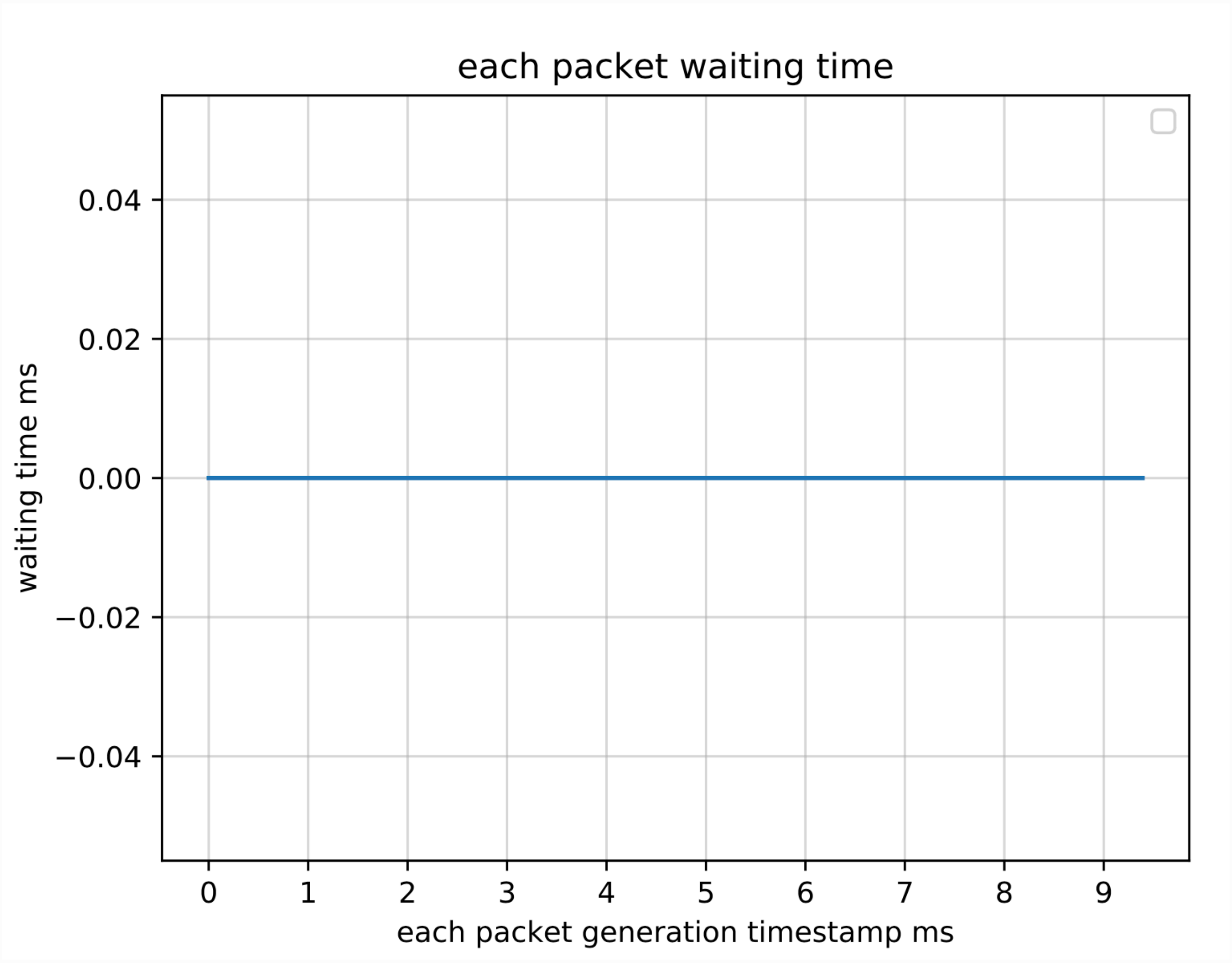
Shell Code

```
python queue_onoff_traffic.py -S 200 -T 10 --on_period 0.3 --off_period 1 --
no-trace >> eachpacket_waitingtime_data200.out
```

Result

```
Each packet delay time:
msg ctime : 0.0000E+00, msg delay : 0.0000E+00
msg ctime : 1.0000E-02, msg delay : 0.0000E+00
msg ctime : 2.0000E-02, msg delay : 0.0000E+00
msg ctime : 3.0000E-02, msg delay : 0.0000E+00
msg ctime : 4.0000E-02, msg delay : 0.0000E+00
msg ctime : 5.0000E-02, msg delay : 0.0000E+00
msg ctime : 6.0000E-02, msg delay : 0.0000E+00
msg ctime : 7.0000E-02, msg delay : 0.0000E+00
msg ctime : 8.0000E-02, msg delay : 0.0000E+00
msg ctime : 9.0000E-02, msg delay : 0.0000E+00
msg ctime : 1.0000E-01, msg delay : 0.0000E+00
msg ctime : 1.1000E-01, msg delay : 0.0000E+00
msg ctime : 1.2000E-01, msg delay : 0.0000E+00
msg ctime : 1.3000E-01, msg delay : 0.0000E+00
msg ctime : 1.4000E-01, msg delay : 0.0000E+00
msg ctime : 1.5000E-01, msg delay : 0.0000E+00
msg ctime : 1.6000E-01, msg delay : 0.0000E+00
msg ctime : 1.7000E-01, msg delay : 0.0000E+00
msg ctime : 1.8000E-01, msg delay : 0.0000E+00
msg ctime : 1.9000E-01, msg delay : 0.0000E+00
msg ctime : 2.0000E-01, msg delay : 0.0000E+00
msg ctime : 2.1000E-01, msg delay : 0.0000E+00
msg ctime : 2.2000E-01, msg delay : 0.0000E+00
msg ctime : 2.3000E-01, msg delay : 0.0000E+00
msg ctime : 2.4000E-01, msg delay : 0.0000E+00
msg ctime : 2.5000E-01, msg delay : 0.0000E+00
msg ctime : 2.6000E-01, msg delay : 0.0000E+00
msg ctime : 2.7000E-01, msg delay : 0.0000E+00
msg ctime : 2.8000E-01, msg delay : 0.0000E+00
msg ctime : 2.9000E-01, msg delay : 0.0000E+00
msg ctime : 1.3000E+00, msg delay : 0.0000E+00
msg ctime : 1.3100E+00, msg delay : 0.0000E+00
msg ctime : 1.3200E+00, msg delay : 0.0000E+00
msg ctime : 1.3300E+00, msg delay : 0.0000E+00
msg ctime : 1.3400E+00, msg delay : 0.0000E+00
msg ctime : 1.3500E+00, msg delay : 0.0000E+00
msg ctime : 1.3600E+00, msg delay : 0.0000E+00
msg ctime : 1.3700E+00, msg delay : 0.0000E+00
msg ctime : 1.3800E+00, msg delay : 0.0000E+00
msg ctime : 1.3900E+00, msg delay : 0.0000E+00
msg ctime : 1.4000E+00, msg delay : 0.0000E+00
msg ctime : 1.4100E+00, msg delay : 0.0000E+00
msg ctime : 1.4200E+00, msg delay : 0.0000E+00
msg ctime : 1.4300E+00, msg delay : 0.0000E+00
msg ctime : 1.4400E+00, msg delay : 0.0000E+00
msg ctime : 1.4500E+00, msg delay : 0.0000E+00
msg ctime : 1.4600E+00, msg delay : 0.0000E+00
msg ctime : 1.4700E+00, msg delay : 0.0000E+00
msg ctime : 1.4800E+00, msg delay : 0.0000E+00
msg ctime : 1.4900E+00, msg delay : 0.0000E+00
msg ctime : 1.5000E+00, msg delay : 0.0000E+00
msg ctime : 1.5100E+00, msg delay : 0.0000E+00
msg ctime : 1.5200E+00, msg delay : 0.0000E+00
msg ctime : 1.5300E+00, msg delay : 0.0000E+00
msg ctime : 1.5400E+00, msg delay : 0.0000E+00
msg ctime : 1.5500E+00, msg delay : 0.0000E+00
msg ctime : 1.5600E+00, msg delay : 0.0000E+00
msg ctime : 1.5700E+00, msg delay : 0.0000E+00
msg ctime : 1.5800E+00, msg delay : 0.0000E+00
Average waiting time:
packet size:2.0000E+02 , average delay:0.0000E+00
```

Diagram



As we can see from the result, the on-off traffic pass through the TBF without shaping.

This is because the token have been generated enough to during the off-period.

Other attempt

Another method without shaping

Another method to avoid shaping is to make capacity big enough or the packet size small enough. it can solve this question to some extend. However It is not a good method in reality.

Shell code

```
python queue_onoff_traffic.py -S 20 -T 2 --no-trace >>
eachpacket_waitingtime_data200.out
```

Result

```
Average waiting time:
packet size:2.0000E+01 , average delay:0.0000E+00
```

It is not a good method.

One experiment with shaping

I am interested in the situation with shaping. Therefore, I implement it below.

Shell Code

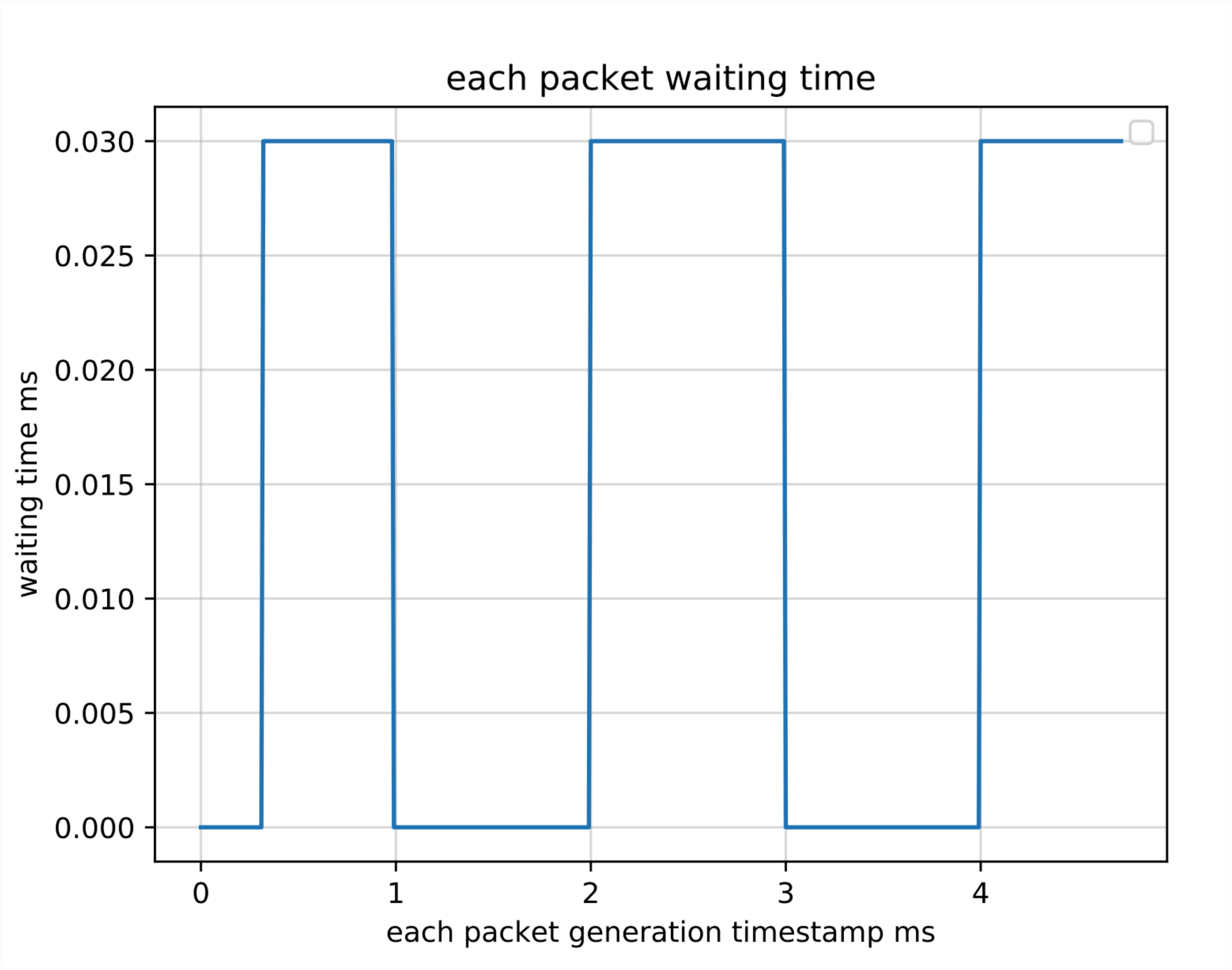
```
python queue_onoff_traffic.py -S 200 -T 10 --no-trace >>
eachpacket_waitingtime_data_shaping.out
```

Result

```
0.0000E+00,0.0000E+00
1.0000E-02,0.0000E+00
2.0000E-02,0.0000E+00
3.0000E-02,0.0000E+00
4.0000E-02,0.0000E+00
5.0000E-02,0.0000E+00
6.0000E-02,0.0000E+00
7.0000E-02,0.0000E+00
8.0000E-02,0.0000E+00
9.0000E-02,0.0000E+00
1.0000E-01,0.0000E+00
1.1000E-01,0.0000E+00
1.2000E-01,0.0000E+00
1.3000E-01,0.0000E+00
1.4000E-01,0.0000E+00
1.5000E-01,0.0000E+00
1.6000E-01,0.0000E+00
1.7000E-01,0.0000E+00
1.8000E-01,0.0000E+00
1.9000E-01,0.0000E+00
2.0000E-01,0.0000E+00
2.1000E-01,0.0000E+00
2.2000E-01,0.0000E+00
2.3000E-01,0.0000E+00
2.4000E-01,0.0000E+00
2.5000E-01,0.0000E+00
2.6000E-01,0.0000E+00
2.7000E-01,0.0000E+00
2.8000E-01,0.0000E+00
2.9000E-01,0.0000E+00
3.0000E-01,0.0000E+00
3.1000E-01,0.0000E+00
3.2000E-01,3.0000E-02
3.3000E-01,3.0000E-02
3.4000E-01,3.0000E-02
3.5000E-01,3.0000E-02
3.6000E-01,3.0000E-02
3.7000E-01,3.0000E-02
3.8000E-01,3.0000E-02
3.9000E-01,3.0000E-02
4.0000E-01,3.0000E-02
4.1000E-01,3.0000E-02
4.2000E-01,3.0000E-02
```

```
4.3000E-01,3.0000E-02
4.4000E-01,3.0000E-02
4.5000E-01,3.0000E-02
4.6000E-01,3.0000E-02
4.7000E-01,3.0000E-02
4.8000E-01,3.0000E-02
4.9000E-01,3.0000E-02
5.0000E-01,3.0000E-02
5.1000E-01,3.0000E-02
5.2000E-01,3.0000E-02
...
```

Diagram



At first, the packet do not need to wait for token because there are enough token in capacity for packet to pass through directly. However,when the token have been consumed fully. packets must wait for enough token generation. Because we have removed the transmission time, the packet must wait for $\frac{messagesize}{tokengernerationrate}$ time forever.

Conclusion

As we can see from the simulation result, the on-off traffic pass through the TBF without shaping.

This is because the token have been generated enough to during the off-period.