Meet Deadline Requirements Challenge

Spring 2022 CS3570 Introduction to Multimedia

Group 20
109062103 林晨 109062206 張雅涵 109062211 張惇媛

ABSTRACT

In multimedia streaming, whether a packet is sent in time is important for users' experience. For the purpose of increasing users' experience, we implement a scheduling algorithm to decide the block sending sequence because of the limit of bandwidth. According to the network condition, we also adjust the sending rate and use a congestion window to optimize bandwidth usage and to avoid packet loss. In "Meet Deadline Requirements Challenge", we use QoE(quality of experience) to determine the performance of our project. After multiple times of modification of our implementation, we found our best solution.

1 INTRODUCTION

In this challenge, we are encouraged to implement a block scheduling algorithm and a bandwidth estimating algorithm. In the provided template, which we used to implement our project, the method – "select_block" is used for block scheduling. "select_block" takes current time and a queue of waiting-to-send blocks as parameters, then returns the index of the block that is best to be sent first. The "cc_trigger " method is used for bandwidth estimating, which is called when the sender receives a packet from its receiver. "cc_trigger" takes the current time and an event information table of the received packet as parameters, estimates the current network condition, then returns the best send rate and congestion window size.

In our project, we implemented these two methods by analyzing the parameters, finding useful data and designing ways to manipulate them using our knowledge of multimedia data transferring, and by referencing some of the demo solutions [1]. With several times of QoE testing and modifying, we finally found the best solution that can most significantly improve the overall performance.

2 BLOCK SCHEDULER

In the select_block method, we determine the mathematical form and what block data should be used in our algorithm by testing QoE. After trying many different data, we eliminate data that does not affect our result such as split_nums. Finally, we decide to use priority, deadline, and size of each block to be the factors of our implementation. We assign a score to each of the factors, and try to make the scores range from 0 to 100.

2.1 Factors

Control should take priority over audio and video, and audio should take priority over video. By testing different values, we finally assigned the priority score of control, audio, and video as 100, 80, and 10 respectively.

For the deadline score, We think if the ratio of the remaining available time for sending a block to the deadline is smaller, the block should be sent first. Since the remaining available time is equal to create time plus deadline minus current time, we have the following equation:

$$100 * \left[1 - \frac{create\_time + deadline - cur\_time}{deadline}\right]$$

When current time and create time are the same, the above value should be 0. However, because of the floating point error of python, the value may be negative. Therefore, we have the following equation to make sure the deadline score is not negative.

$$score_{deadline} = \max(0, 100 * [1 - \frac{create\_time + deadline - cur\_time}{deadline}])$$

For the size score, since we do not know the maximum and minimum value of block size, without making the score range from 0 to 100, we use the block size as the score directly, and hence the value may be very large.

2.2 Block Scoring Function

We design two kinds of block scoring functions. The first one is a linear function as shown below.

$$score = p \times score_{priority} + d \times score_{deadline}$$

We don't use size score here because it is hard to determine the coefficient of size score since we don't know the range of it.

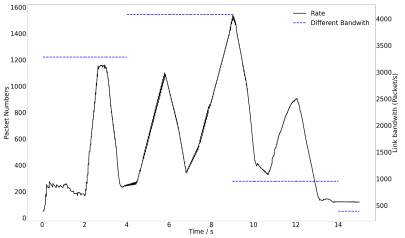
The second block scoring function is a nonlinear function. We think a block with larger size should be sent later, so the power of size is set negative which means size score is the denominator.

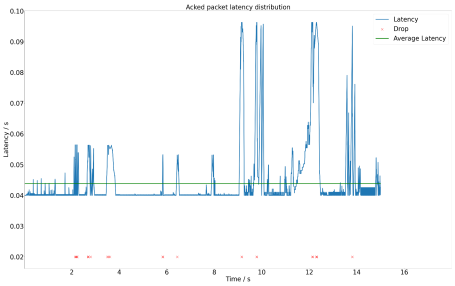
Therefore, we have the equation shown below.

$$score = \frac{score_{priority}{}^{p} \times score_{deadline}{}^{d}}{score_{size}{}^{s}}$$

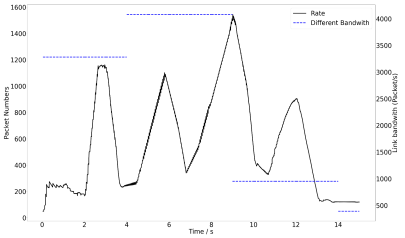
We tried several different values of the three factors' parameters(p, d, and s) with only one selected dataset, and recorded QoE and the output figures of changing rate and emulator-analysis as shown below. The upper figure is for changing rate and the lower figure is emulator-analysis.

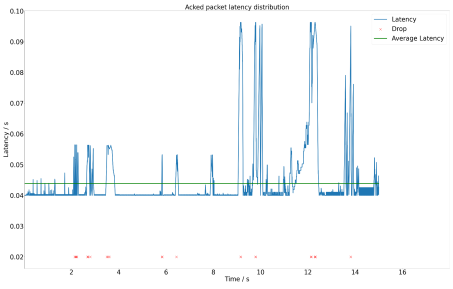
Initial output(without implementing block scoring function). QoE = 196.



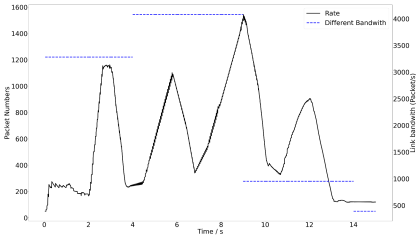


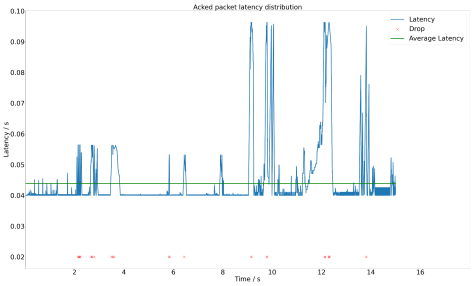
Using linear function and (p, d) = (0.5, 0.5). QoE = 496.



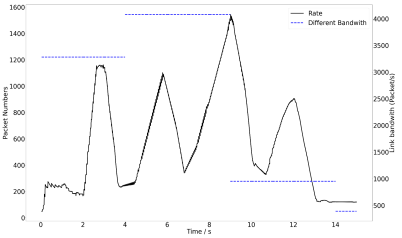


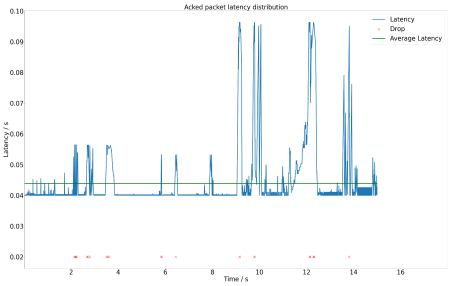
Using linear function and (p, d) = (0.7, 0.3). QoE = 506.



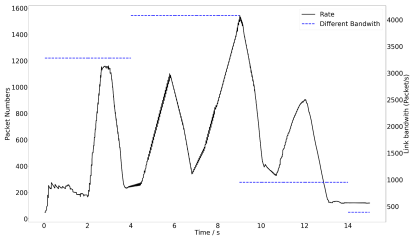


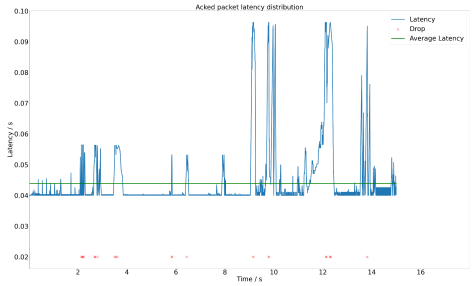
Using linear function and (p, d) = (0.9, 0.1). QoE = 506.



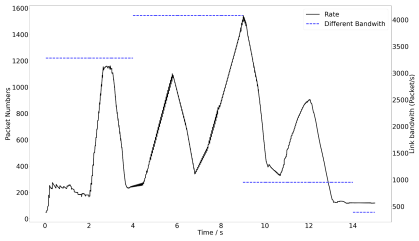


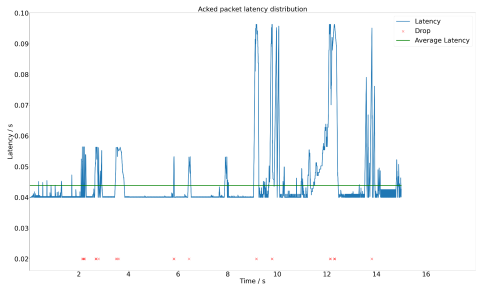
Using nonlinear function and (p, d, s) = (2, 0.1, 2). QoE = 518.



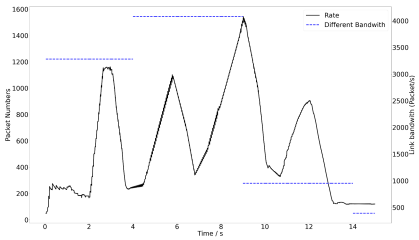


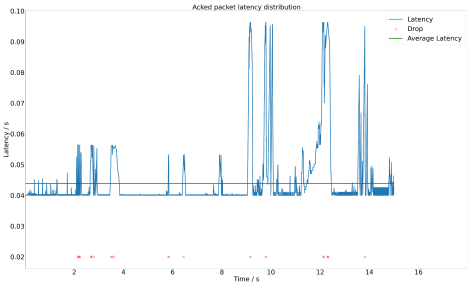
Using nonlinear function and (p, d, s) = (1, 1, 1).
QoE = 519.





Using nonlinear function and (p, d, s) = (2.5, 0.2, 0.05).
QoE = 531.





Based on the observation, we determine to use a nonlinear function with (p, d, s) = (2.5, 0.2, 0.05). Hence, the final block scoring function is shown below.

$$score = \frac{score_{priority}{}^{2} \times score_{deadline}{}^{0.02}}{score_{size}{}^{0.05}}$$

3 BANDWIDTH ESTIMATOR

In the cc_triger method, we control the sending rate and congestion window size. We first invoke a new method called “estimate_bandwidth” from the provided solution of PyTorch, which uses reinforcement learning to adjust sending rate. For the congestion window control, we implemented the TCP Reno method and modified the action of fast recovery state.

3.1 SENDING RATE

In the estimate_bandwidth method, we count the received packet numbers, 50 as a round. We only take action(return a new send rate) on receiving the 50th packet, and start another round of counting, this way we can avoid taking wrong actions on some packet event that only happened temperately.

In our solution class, we created lists that keep track of all the previous packets’ information, for example we have a “result_list " that records the event type(loss or ACKed) of each received packet. Every time the estimate_bandwidth method is invoked, we append the information such as event type, latency, …etc. into these lists. With these information, it is now possible to estimate the loss rate and throughput of the network, which can be used to indicate network condition.

Every time we count 50 packets sent, we apply the DQN algorithm to find the best way to adjust the send rate according to the current estimated network condition. The input data includes – send_rate/maximum bandwidth, sum loss rate, instant loss rate, sum rate, instant rate, latency

list(50 elements, refreshed every round), last action, last loss rate, and last sum loss rate…etc. The output is one of the three actions – increase the sending rate (current sending rate times 1.4), keep the current sending rate, and decrease the sending rate (current sending rate times 0.4).

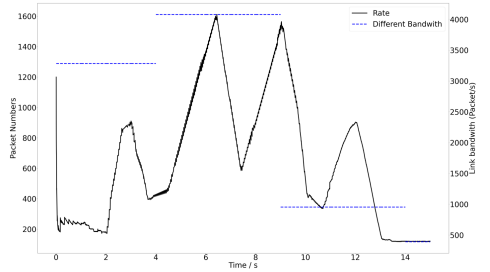
3.2 CONGESTION WINDOW

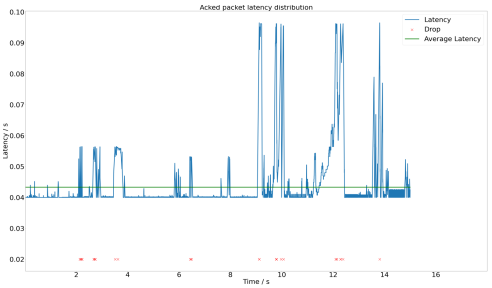
We implement the method of TCP Reno. In TCP Reno, after a packet loss event, it will enter the fast recovery state, where the size of the congestion window drops to half of the threshold. According to our observation, in temporary congestion, if the drop rate of the congestion window is high, it has to spend lots of time on increasing the congestion window. In contrast, in long-time congestion, if the drop rate of the congestion window is low, it has to spend lots of the time on decreasing the congestion window. As a result, we modify the fast recovery state to make the congestion window drop to the threshold divided by 2.4 when larger than two packets lost in five instant packets. Also, if the size of the congestion window is low, the more timeout events will occur and result in lower QoE. We set the minimum size of the congestion window to ensure that the transmission rate is not too low.

3.3 RESULT

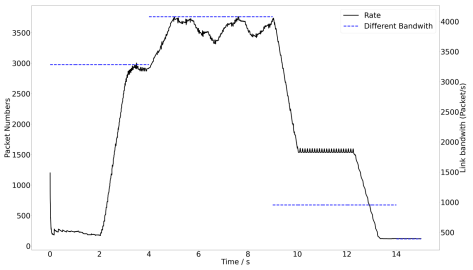
We tested different sets of size of the congestion window for one selected dataset. The output pictures are as shown below.

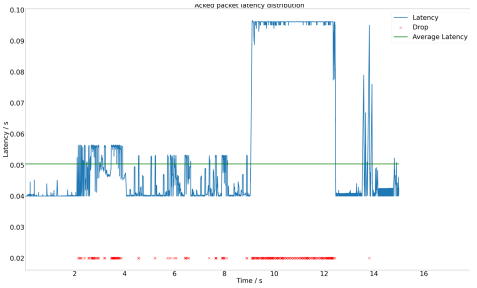
init_cwnd = 10 and min_cwnd = 10, QoE = 524.





init_cwnd = 150 and min_cwnd = 150, QoE = 552.





4 CONCLUSION

In this project, we modified the block scheduler and bandwidth estimator by using different sets of values for power of block scheduler's factors, minimum sending rate, minimum congestion window size, threshold decreasing portion…etc. based on our knowledge of multimedia transfer, but soon we reached a limit. After applying a reinforcement learning model, we reached a higher QoE.

For improvement, we may put efforts to make the sending rate and congestion window more flexible to adjust in a short period of time based on some rapidly changing network conditions.

REFERENCE

[1] Details of ACM Multimedia 2021 Grand Challenge: Meet Deadline Requirements https://github.com/AItransCompetition/Meet-Deadline-Requirements-Challenge/tree/master/solution_demos

[2] Introduction of TCP Tahoe and TCP Reno https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno/