Simulations of Flow Fair Scheduling under Data Traces

CMU

August 29, 2019

1 Problem review

We still consider the six-node network, shown in Figure 1. The only differences between this set of simulations and the previous simulations for this network are: we extract the empirical probability transition matrix from the background traffic data, instead of using a uniform distribution; we simulate the random bandwidth utilization weight of each protocol by leveraging the rtt data of protocols Reno and Cubic as well as the background traffic data, instead of using uniform distributions. Details can be found in the next section.

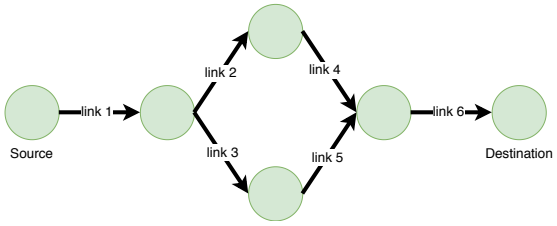

Figure 1: Network topology used in simulations

2 Data Processing

The code for processing the data can be found at `https://github.com/ChelseaZhang90/gym-flowsched/blob/master/samples/data_pre-process.ipynb`.

More specifically, we first derive the total available bandwidth capacity of our flows in each timestep by subtracting the background traffic in each timestep from the maximum background traffic over all timesteps. Then, we fit the resulted data of available bandwidth of our flows into a 20-point distribution. We write the 20-dimensional vector into file **state_dist.txt**. We then define the probability transition matrix in our environment (`https://github.com/ChelseaZhang90/gym-flowsched/blob/master/gym_flowsched/envs/flowsched_data_env.py`) to be the vector in **state_dist.txt**.

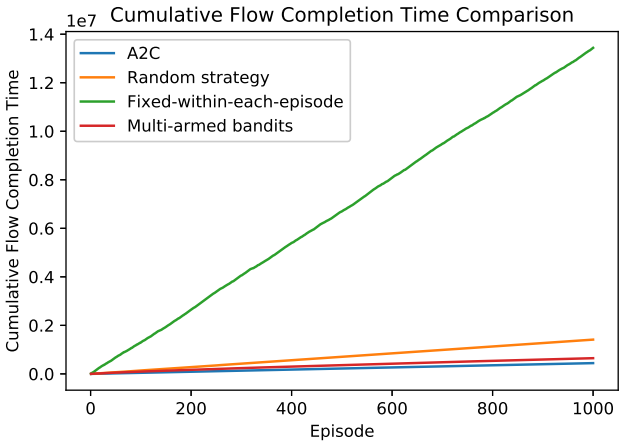

Figure 2: A2C achieves lower total flow-time using data traces.

For the random weights, we first compute the weight of each protocol (Reno and Cubic) in each timestep by the following equation

$$\text{weight} = \frac{\text{Packet size}}{\text{Rtt per packet} \times \text{Average bandwidth capacity of our flows}}$$

We then fit the data of weights of Reno and Cubic into two distributions respectively, by using the kernal density estimation function supported by Scipy (cf. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html`) and write the supports and probability density (pdf) of the two distributions into files **reno_wt_supports.txt**, **cubic_wt_supports.txt**, **reno_wt_pdf.txt**, and **cubic_wt_pdf.txt**, respectively. We also define an additional protocol, the weight of which follows a normal distribution with an expectation of 0.3. Finally, we draw the weight of each protocol in each timestep from the corresponding distribution so that to determine the transmission rate achieved by each protocol.

3 Algorithm Performance

With gym and our environment installed, the main function (`https://github.com/ChelseaZhang90/gym-flowsched/blob/master/samples/run_multi_path.py`) can be run by command: **make normalData** (cf. gym-flowsched/samples/Makefile). As shown in Figure 2, our reinforcement learning algorithm A2C outperforms than the other benchmark strategies.