RL Framework

Zhao Xiaoyang

March 25, 2019

1 Environment

The Internet consists of several autonomous systems(ASes), each of them being a network (or domain) and interacting with others in the process of business negotiation and traffic management. Traffic flows from source to specific destination are forwarded across the Internet under cooperation of related autonomous systems. Autonomous system uses BGP, the only exterior gateway routing protocol, to choose appropriate AS-path for by-passing flows. In this *distance vector route* process, on the one hand, autonomous system adopts AS-path choosing strategy based on locality information. Each autonomous system has business property, which means they are not fully cooperative with each other when transmitting flows. For example, some link statuses among autonomous systems and routing strategy settings need to be hidden. On the other hand, autonomous systems are selfish to achieve higher own profits(e.g., minimize price cost for transmitting, maximize throughput of by-passing flows). They are not likely to consider load-balance of the Internet using BGP by default.

We consider the Internet as a multi-agent system, each autonomous system is an agent with route choosing strategy. The strategy follows basic rules of business relationship: (i) prefer customer ASes as next-hop to peer ASes; (ii) prefer peer ASes as next-hop to provider ASes. Distributed reinforcement learning framework is run on each agent to learn a better route strategy for incoming traffic matrix. Learning and executing process of each agent are fully distributed without a special-designed centralized agent.

The training process of reinforcement learning is divided into time rounds. At each round, there is a traffic matrix(several flows) with source and destination information to be transmitted. Based on current strategy, each agent chooses next-hops to forward flows. Choosing action for each flow at the same time will cause an exponentially growing action space. Therefore, we sequencelly inject flows from traffic matrix: Firstly inject one flow from traffic matrix, each agent takes action to choose next-hop for it; then routing path of this flow is determined, network status(e.g., link load, flow throughput) is updated; then inject another flow, repeat above steps until all flows in traffic matrix are injected; finally, network gets the final new status for this round, agent gets feedback(reward) from network environment to improve strategy.

We consider the objective of the whole network system as the sum of flows'(in traffic matrix) end-to-end throughput. This is because RL framework is designed as part of path-selection in BGP protocol, which is run on the upper level of TCP protocol. Strategies of agents not only cooperatively route incoming flows, but also competitively maximize end-to-end throughput of their own by-passing flows, respecting to the limited capacity of network links.

2 Settings

Here we detail the design of reinforcement learning module on each agent. The network topology abstracted has N agents(nodes), E links(edges) and capacity C_e for link $e \in E$.

State space. At each time round, there is a 0-1 $N \times N$ traffic matrix where element (f_s, f_d) with value 1 represents incoming flow f with source f_s and destination f_d. Observation of agent i is $o_i = (f_s, f_d, \vec{h}, \vec{v})$, includes the following information:
(1) f_s, f_d is source and destination agent of the incoming flow. We assume that flow profile can be known to any by-passing agent(autonomous system). Taking this as input can help agent output action for specific flow.
(2) $\vec{h}$, an dim_h-dimensional vector where dim_h is the number of neighboring(directly linked) agents. Each item in the vector represents traffic load status of the link with neighboring agent. This is

because autonomous system is local observational, neighboring links is the most directly knowledge it can based on to choose next-hop.
(3) $\vec{v}$, an dim_v-dimensional vector where dim_v is the number of flows passing this agent last round. Each item in the vector represents end-to-end throughput of one flow. The rationale behind this is that routing policy is run on the upper level of TCP protocol, and using end-to-end throughput can help agent capture link status far beyond neighboring observation, which is useful for network load-balancing. Besides, it is realistic for agent(autonomous system) to obtain end-to-end throughput of by-passing flows.

Based on observations of all agents, we define three types of states under different situations:
(1) $s^i_{glb} = (o_k, a_{kj})_{k \in N, j \in F}$, where a_{kj} represents agent k's action for flow j(in traffic matrix). Global state supposes that learning agent i knows the actions and observations of all other agents. This is not realistic for inter-domain routing because of selfishness and locality of autonomous system, but can be an upper bound for learning performance. The rationale is that, on the one hand, combination of other agents' observations helps learning agent to have a maximum view of the whole network; on the other hand, combination of taken actions makes the environment stationary even as the policy of each agent changes.
(2) $s^i_{nei} = (o_k, a_{kj})_{k \in M_i, j \in F}$, where M_i represents neighboring agents set(include itself) of learning agent i. Comparing with global state, neighboring state is more realistic and extends agent's view.
(3) $s^i_{loc} = (o_i)$. Local state represents that learning agent i only has observation itself. Every agent takes action independently and regards others as part of the environment, causing environment to be unstationary; besides, because of the uncertainty of other agents' actions, it's not a Markov process for agent to step into next state(the same input for RL module can cause different output).

The input state space to reinforcement learning model can be any of three types above, based on specific property of autonomous system.
Action space. After receiving s_i, learning agent i selects an action a_i based on a policy $\pi^i_\theta(s_i, a_i)$, which is a probability distribution over the action space. The policy is produced by NN with θ as the set of parameters. Naturally, an agent can transmit different flows in one round. For flows with different destinations information, candidate next-hop set can be different because of BGP routing advertisement. Considering action space as a combining of all flows'(in traffic matrix) next-hop set will cause size of action space to grow exponentially. To reduce action space and expedite policy learning, we simplify the action space definition A_i and each action is a forwarding next-hop from neighboring agents set M_i. In this way, the size of action space can be really small, only be the value of agent's neighboring-agents number(degree).

To satisfy BGP advertising rules, in the output layer of policy network, we mask the invalid actions, which take agent not existing in routing table's AS path as next hop, by setting its probability to 0 in the policy distribution. The we rescale the probabilities on all actions such that the sum still equals to 1.
Reward. We target to maximize the sum of end-to-end throughput of traffic matrix. However, each agent is selfish and aims to maximize their own rewards. We define the reward for each agent as the average end-to-end throughput of flows it transmits in this round: $r_i = \frac{\sum v}{n_i}$, where n_i represents the number of by-passing flows and $\sum v$ is sum of their end-to-end throughput. As each link between agents has a utility capacity, maximizing own reward for each agent is a competitive process. Besides, global state s_{glb} and neighboring state s_{nei} are designed to enhance cooperation among agents. Hence, the objective of the whole network system are based on mixed environment.
Techniques. We adopt a number of techniques to stabilize training, expedite policy convergence and improve the quality of learned policy:
(1) *Actor-Critic framework.* The basic policy gradient training for each agent is to maximize the expected cumulative discounted reward $J(\theta) = E[\sum_{t \geq 0} \gamma^t r_t]$, where $\gamma \in [0, 1]$ is the discount factor. The policy gradient used for NN update can be calculated as:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\sum \nabla_\theta log(\pi_\theta(s_i, a_i)) Q^{\pi_\theta}(s_i, a_i)]$$

where the Q value indicates the 'quality' of action a_i under s_i for learning agent i, calculated as expected cumulative discounted reward obtain after selecting a_i under s_i following π_θ.

However, high variance in Q values prevents convergence of the policy model. *Actor-critic* algorithm introduces an advantage function, i.e., $Q^{\pi_\theta}(s_i, a_i) - V^{\pi_\theta}(s_i)$, where $V^{\pi_\theta}(s_i)$ calculated as the expected cumulative reward following the policy π from state s, over all possible actions in the state. Then this advantage function is used for policy gradient calculation instead of Q value.

Specifically, the *Critic* means a value network that trained using temporal difference method to estimate $V^{\pi_\theta}(s_i)$. The *Actor* is a policy network using 'guidance' from output of *Critic*, then outputs action distribution over action space. Both *Actor* and *Critic* consists of three layers: input, hidden(fully connected) and output.
(2) *Decaying exploration.* In order to perceive more situations under the network environment, we use $\epsilon - greedy$ to ensure that the action space is adequately explored; or the system may converge to a poor local optimal stage. To be specific, each time round, each agent has probability ϵ to randomly choose a valid action from action space for each flow; has probability $1-\epsilon$ to choose action following current policy π_θ. Besides, we gradually decay the value of ϵ after constant rounds. This is because after enough time rounds, too much unnecessary exploration may prevent convergence of NN model.
(3) *Concurrent experience replay.* We adopt experience replay to alleviate correlation in the sample sequence. The rationale behind this is that training with consecutive samples may be hard to converge. For example, the high reward after executing one time round will cause agents to adopt action following the same policy, which prevent agent from exploring samples with higher reward. Besides, in a multi-agent system, we store all agents' samples(s, a, s', r) in the concurrent round together as a group into a FIFO replay buffer with a fixed size. Then in the training process, we select several groups from replay buffer as training samples, as if each group with multi-agent is executing concurrently.