Iroko: A Framework to Prototype Reinforcement Learning for Data Center Traffic Control

• Fabian Ruffy
• Michael Przystupa
• Ivan Beschastnikh

University of British Columbia, Canada
Objective:

- Overcome difficulties of Reinforcement Learning to make it useful to learn optimal network policies.
- Design an emulator which allows researchers to deploy different networking topologies and evaluate different congestion control algorithms.

Problem Definition

- Identify difficulties faced by RL algorithms.
- Analyze requirements for Reinforcement Learning to succeed in the datacenter context
Motivation to use RL in networking

• Many data center networking challenges can be formulated as RL problems.
• Some of the problems include: Data-driven flow control, routing and power management.
• RL has the objective of maximizing future rewards.
• RL models have the capability to learn anticipatory policies.
• Current policies are mostly reactive which respond to micro-bursts and flow-collisions.
Difficulties in using Reinforcement Learning

• RL algorithms often suffer from overfitting.
• RL researchers can try out unlimited environmental state representations which can cause RL models to overfit.
• RL algorithms lack reproducibility.
• Reproducibility can be affected by extrinsic factors (e.g. hyperparameters or codebases) and intrinsic factors (e.g. effects of random seeds or environment properties).
• Data center operators expect stable, scalable and predictable behavior.
Requirements of RL

Patterns in Traffic:
• PCC and Remy are two techniques that demonstrate that congestion control algorithms can be evolved from trained data.
• DC traffic pattern can be used to design a proactive algorithm which forecasts traffic matrix and controls host sending rates.

Centralized control algorithms:
• Centralized policy has global view.
• It has ability to plan ahead and grant hosts traffic rates based on the model.
Requirements of RL

Sources of Information:

• CC algorithms use data from transport layer and below.
• It is possible to collect data from network links, switches and other components of hardware.
• Essential to collect congestion signals.
• Some features: switch buffer occupancy, packet drops, port utilization, active flows, and RTT, latency, jitter and queue length.
• Throughput can be used as a metric to optimize.
• One-hot encoding of active TCP/UDP flows per switch port can be used to identify network patterns.
Emulator Design

Key components:
• Network topologies
• Traffic generators
• Monitors
• Agents to enforce congestion policy

Mininet: Software Defined Networking Simulator that can run on single laptop.

RLlib: Library that provides RL abstractions like defining policy, optimizer etc.
Figure 1: Architecture of the Iroko emulator.
RL implementation in Iroko

Agent action:

• We represent this action set as a vector 'a' of dimensions equal to the number of host interfaces.

• Each dimension $a_i$ represent % of max bandwidth allocated.

\[ bw_i \leftarrow bw_{max} \times a_i \quad \forall \quad i \in \text{hosts} \]

Reward Function:

\[ R \leftarrow \sum_{i \in \text{hosts}} \left( \frac{bw_i}{bw_{max}} \right) - \text{ifaces} \cdot \left( \frac{q_i}{q_{max}} \right)^2 - \text{std} \]

\text{bandwidth reward} \quad \text{weight} \quad \text{queue penalty} \quad \text{devpenalty}
Experiments

• Compare the performance of 3 RL algorithms with TCP New Vegas and DCTCP.

• DCTCP: Switches mark packets after the queue length exceeds a threshold.

• TCP New Vegas: Changes the congestion window size based on the RTT observed in packages.

• Rewards for TCP algorithms are also calculated.

• TCP's CC can be confounding with RL's CC
Results
Conclusion

• Great contribution towards Machine Learning: Interfaced with OpenAI gym

• Carefully analyzed the requirements for RL and tried to provide them in the framework.

• Enables researchers to see the performance of conventional non-RL algorithms through the lens of reward function.

• Not specified the nature of hardware simulated.

• Deals with protocols from TCP/IP stack.
Overview of RL
Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a | \theta^Q)$ and actor $\mu(s | \theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$

for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for $t = 1, T$ do
        Select action $a_t = \mu(s_t | \theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
        $$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu)|_{s_i}$$
        Update the target networks:
        $$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$
        $$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$
    end for
end for
Overview of RL Methods

- [https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287](https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287)
- [https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419](https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419)

- PPO: Standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates.
- Reinforce: Weight adjustments in direction of gradients of immediate reinforcement and delayed reinforcement.