December 06, 2018

A REINFORCEMENT LEARNING (RL) MODEL FOR DYNAMIC OPTIMIZATION OF SWITCH PORT/QUEUE BUFFER ALLOCATION

Imran Pasha

A REINFORCEMENT LEARNING (RL) MODEL FOR DYNAMIC OPTIMIZATION OF SWITCH PORT/QUEUE BUFFER ALLOCATION

AUTHOR:
Imran Pasha

ABSTRACT

Techniques are described herein for a reinforcement learning (RL) model for dynamic optimization of switch port/queue buffer allocation. According to the described techniques, a trained neural network model can be installed on a switch or on a network management server to dynamically adjust the shared/dedicated buffer allocation for the low/high priority queues in case of a frame loss. The dynamic buffer adjustment continues till the port/queue no longer experience a frame loss.

DETAILED DESCRIPTION

Typically, switch buffer management algorithms and templates allocate the per port priority queue(s) buffer at the Line Card boot-up phase. Thereafter, irrespective of the traffic profile observed, the buffer allocation remains unchanged. Consequently, the high and low priority queues suffer from sub-optimal buffer allocation often resulting in frame drops.

The techniques described herein provide a reinforcement learning model for dynamic optimization of switch port/queue buffer allocation. In the Artificial Intelligence/Machine Learning (AI/ML) realm, the problem can be framed as a Reinforcement Learning (RL) Model that interacts in a dynamic environment and takes corrective actions to ensure the high/low priority queue buffer allocation is optimized to minimize the frame loss.

The dynamic environment monitors the buffer allocation, queue occupancy, queue priority, Quality-of Service (QOS) configuration and per port and per queue traffic counters. The environment robustness and sophistication can be enhanced by including more relevant state. The model dynamically adjusts the buffer allocation in reaction to the detection of frame loss. The buffer allocation adjustment continues till the frame loss is

Published by Technical Disclosure Commons, 2018

rectified. The RL model can tap into the available buffer pool or steals from a port/queue that might have excess buffer allocated.

The canonical Q-learning (model free) RL design for the agent and environment interaction is shown in Figure 1 below.

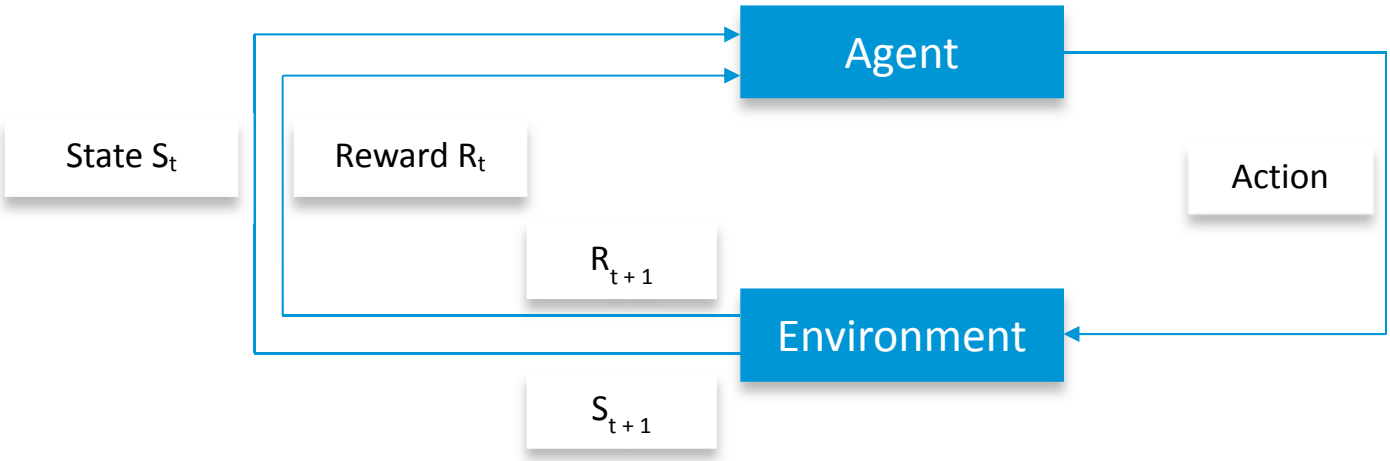

Figure 1

Agent: takes actions **(A)**

Actions (A): Set of possible directives an agent can execute.

1. Increase buffer allocation
2. Decrease buffer allocation
3. No action

Environment: is the continued observed switch state the agent operates on.

1. Input: agent's current state and action
2. Output: agent's reward and next state

Two separate but related environments per slice (SOC) are modeled

1. Shared buffer for all ports, 8 queues/port
2. Dedicated buffer for all port, 8 queues/port

The high priority queue is allocated 10% of the total buffer.

The low priority queue is allocated 90% of the total buffer.

Matrix representation [N x M] = [Port[1x3], Queue[1x3], State[1x8]] = [9 x 8]

State (S): is an immediate situation the agent operates in and reacts to.

0 = Port (P) = 3 per SOC (adjustable to max per SOC, substantially reduces the training time)

1 = Queue (Q) = 3 per port (adjustable to max per SOC, substantially reduces the training time)

2 = Queue minimum threshold (QMin): below this threshold the Q experiences packet drop

3 = Queue maximum threshold (QMax): the maximum buffer that can be allocated to a Q

4 = Queue buffer allocation (QAlloc): the current buffer allocation for a given Q

5 = Port buffer available (PAvail): buffer available for a given Port (P)

6 = Queue packet drop (QDrop): represents whether a Q is experiencing packet drop or not based on the queue occupancy and queue drop counter

(*e.g.*, 0 = no drop, 1 = drop)

7 = Port packet drop (PDrop): represents whether a Port/(any queue) is experiencing packet drop or not based on the per port, per queue drop counter

(*e.g.*, 0 = no drop, 1 = drop)

Reward (R): measures the success or failure of an agent's actions.

1. idx = 9 [0,8]
2. **Action = no action**
    1. (state[idx][2] == state[idx][4]) and (state[idx][6] == 0) and (state[idx][7] == 1)
        1. reward = 1
    2. state[idx][3] == state[idx][4]) and (state[idx][5] >= 0) and (state[idx][6] == 0) and (state[idx][7] == 0)
        1. reward = 3
    3. (state[idx][5] == 0) and (state[idx][6] == 1)
        1. reward = 2
    4. (state[idx][5] == 0) and (state[idx][7] == 0)
        1. reward = 4
    5. else reward = -1

3. **Action = increment QAlloc**
    1. (state[idx][5] > 0) and (state[idx][6] == 1)
        1. reward = 1
    2. (state[idx][3] > state[idx][4]) and (state[idx][5] > 0) and (state[idx][7] == 0)
        1. reward = 2
    3. else reward = -1
4. **Action = decrement QAlloc**
    1. (state[idx][2] < state[idx][4]) and (state[idx][5] >= 0) and (state[idx][7] == 1)
        1. reward = 2
    2. else reward = -1
5. **Any other action**
    1. reward = -100

Discount factor ($\gamma = 0.9$): is multiplied with future rewards to dampen their effect on the agent's choice of action. It makes future rewards worth less than the immediate rewards.

Policy (π): is the strategy employed by the agent to determine the next action based on the current state. It maps states to actions that promise the highest reward.

Q value (Q): maps state, action pairs to the highest combination of immediate reward with all future rewards that might be harvested by the later actions. See Equation 1 below.

$$Q(S_t, a_t) \leftarrow (1 - \alpha) \cdot Q(S_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(S_{t+1}, a_t))$$

$Q(S_t, a_t)$ old value, $\max_a Q(S_{t+1}, a_t)$ estimate of optimal value, $(r_t + \gamma \cdot \max_a Q(S_{t+1}, a_t)$ learned value

Equation 1

Alpha (α): is the learning rate dynamically adjusted by the optimizer function Adagrad

Epsilon (e = 0.25): exploration vs. exploitation factor e(x) := (0, x < 0), (0.25 - x, x >= 0 and x <= 0.25), (0, x > 0.25)

The proposed Q-learning Neural Network model according to the techniques described herein improves the current solutions that deploy buffer allocation templates or static buffer allocation algorithms, therefore, cannot respond to the dynamic changes in the

traffic profile. Specifically, the shared/dedicated buffer management model continuously monitors the port/queue buffer allocation, queue occupancy and packet drop counters, reacts intelligently and re-balances the buffer allocation to eliminate the packet drop condition with > 98 % accuracy. Through training, the model becomes aware of the policy concepts, such as available resource utilization, resource over-allocation, available resource depletion, and release of resource from an over-allocated queue.

A working model in accordance with the techniques described herein is provided. The RL Neural Network model is implemented with Python and Keras library. The initial results produce > 98 % accuracy in re-balancing the buffers. A few example results are cited below. The model environment includes: 3 ports, 3 queues per port, high priority queue buffer allocation 10% (10), low priority queues buffer allocation 90% (18, 1/5 scale factor). The matrix representation [N x M], N = 9, M = 8, labels = [Port, Queue, QMin, QMax, QAlloc, PAvail, QDrop, PDrop]

Example 1: Initial condition: packet loss experienced by all ports/queues

[[0. 0. 0. 10. 0. 28. 1. 1.]
[0. 1. 0. 18. 0. 28. 1. 1.]
[0. 2. 0. 18. 0. 28. 1. 1.]
[1. 0. 0. 10. 0. 28. 1. 1.]
[1. 1. 0. 18. 0. 28. 1. 1.]
[1. 2. 0. 18. 0. 28. 1. 1.]
[2. 0. 0. 10. 0. 28. 1. 1.]
[2. 1. 0. 18. 0. 28. 1. 1.]
[2. 2. 0. 18. 0. 28. 1. 1.]]

Model recommendation: all ports/queues experience no packet loss

[[0. 0. 9. 10. 10. 0. 0. 0.]
[0. 1. 3. 18. 3. 0. 0. 0.]
[0. 2. 15. 18. 15. 0. 0. 0.]
[1. 0. 3. 10. 5. 0. 0. 0.]
[1. 1. 13. 18. 15. 0. 0. 0.]
[1. 2. 5. 18. 8. 0. 0. 0.]
[2. 0. 6. 10. 8. 0. 0. 0.]
[2. 1. 3. 18. 4. 0. 0. 0.]
[2. 2. 15. 18. 16. 0. 0. 0.]]

Example 2: Random initial condition: packet drop at port 0/queue 0, port 0/queue 1, port 1/queue 1 and port 2/queue 1

[[0. 0. 9. 10. 0. 0. 1. 1.]
[0. 1. 15.18. 14. 0. 1. 1.]
[0. 2. 3. 18. 14. 0. 0. 1.]
[1. 0. 2. 10. 10. 0. 0. 1.]
[1. 1. 14.18. 0. 0. 1. 1.]
[1. 2. 4. 18. 18. 0. 0. 1.]
[2. 0. 4. 10. 10. 0. 0. 1.]
[2. 1. 2. 18. 0. 0. 1. 1.]
[2. 2. 16.18. 18. 0. 0. 1.]]

Model recommendation: all ports/queues, no packet loss

[[0. 0. 9. 10. 9. 0. 0. 0.]
[0. 1. 15.18. 15. 0. 0. 0.]
[0. 2. 3. 18. 4. 0. 0. 0.]
[1. 0. 2. 10. 4. 0. 0. 0.]
[1. 1. 14.18. 16. 0. 0. 0.]
[1. 2. 4. 18. 8. 0. 0. 0.]
[2. 0. 4. 10. 8. 0. 0. 0.]
[2. 1. 2. 18. 2. 0. 0. 0.]
[2. 2. 16.18. 18. 0. 0. 0.]]