Let's Play Congestion Control*

Extended Abstract†

Maximilian Bachl
Institute of Telecommunications
Vienna, Austria
maximilian.bachl@tuwien.ac.at

Tanja Zseby
Institute of Telecommunications
Vienna, Austria
tanja.zseby@tuwien.ac.at

Joachim Fabini
Institute of Telecommunications
Vienna, Austria
joachim.fabini@tuwien.ac.at

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings.[1]

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Put some comma-separated keywords here

ACM Reference Format:
Maximilian Bachl, Tanja Zseby, and Joachim Fabini. 2017. Let's Play Congestion Control: Extended Abstract. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.475/123_4

1 PARTIAL ACTION ACTOR CRITIC LEARNING

Partial Action Actor Critic Learning (PAL) is a modification of the Asynchronous Advantage Actor Critic algorithm proposed by Mnih et al. [1]. It consists of two Artificial Neural Networks (ANNs), the Actor Network and the Value Network (the Critic part in the abbreviation stands for the Value Network). Given a state, the Actor Networks outputs what it deems to be the optimum action to perform in that certain state. The Value Network estimates what long-term reward can be expected in this state. So an action is considered good if it achieved a long-term reward that is higher than the long-term reward expected by the Value Network and it is considered bad if the reward was lower than expected. The long-term reward is implemented as a moving average of future rewards. So if a high reward can be achieved right now this is more favorable than if it can be achieved in the future. However it can also be beneficial to get a low reward now and instead get a very large one in the future.

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document
[1]This is an abstract footnote

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

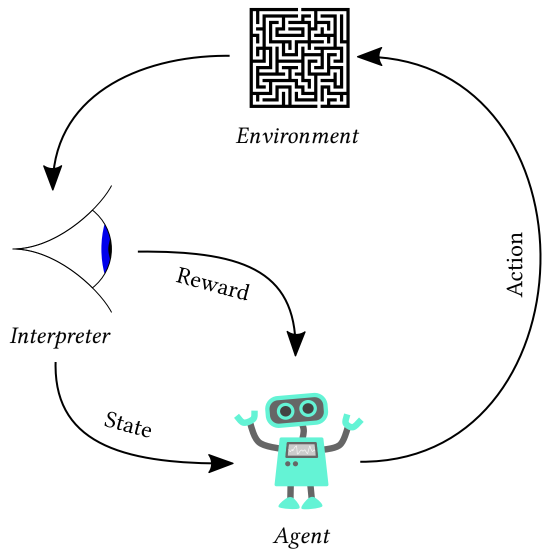

Figure 1: The classical reinforcement learning approach.[a]

[a]adapted from https://upload.wikimedia.org/wikipedia/commons/1/1b/Reinforcement_learning_diagram.svg

1.1 Algorithm

In Algorithm 1 we show the code that runs in one of the learning threads. As proposed in literature, we use several concurrent threads to improve training performance. A major difference to previous approaches to reinforcement learning is the fact that we allow more actions to be taken before the previous actions' rewards have been received. In classical reinforcement learning, an action is always followed by a reward and a reward is always followed by a action. In our proposed concept, however, it is possible to take new actions while the previous action hasn't received its reward yet.

Another major difference in PAL is that one action generates a number of partial actions (≥ 0) (see Figure 2). Each partial action generates feedback upon interacting with the environment. Upon receiving feedback for a partial action, the agent determines the current state and triggers a new action. When all partial rewards of one action were received, the agent combines them to form the reward and updates the value and actor networks (see Algorithm 2).

1.2 Value Network

The value network estimates the expected long-term reward given a state s_t.

Let r_t be the reward that was received at time t. $V(s_t; \theta_v)$ designates the expected mean reward in state s_t given the parameters

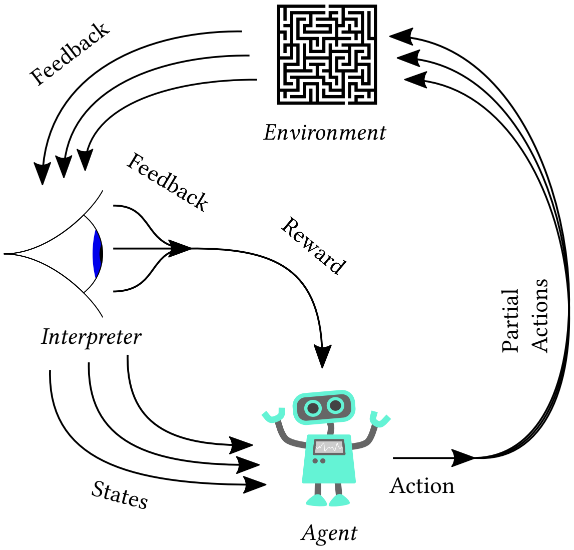

Figure 2: Partial Action Actor Critic Learning: An action consists of zero or more partial actions which trigger partial rewards upon interacting with the environment. Each partial reward updates the state. The value and actor networks are updated upon receiving all partial rewards of one action.[a]

[a]adapted from https://upload.wikimedia.org/wikipedia/commons/1/1b/Reinforcement_learning_diagram.svg

(neural network weights) of the value network θ_v. Then, with γ being the roll-off factor, which designates the influence that future reward has on the moving average, (set to 0.99), we define the average reward as

$$R_t = \left(\left(\sum_{i=0}^{k-1} \gamma^i r_{t+i} \right) + \gamma^k V(s_{t+k}; \theta_\text{v}) \right),$$

where k is upper-bounded by t_max (t_max is a fixed hyperparameter that indicates how many steps should be performed before updating the neural network). So R is simply a moving average of rewards. However, usual moving averages take into account values from the past while this one uses values from the future; it runs reversely.

The loss function, which the value network tries to minimize, is the square of the difference of the actual average reward received and the expected average reward

$$l_{\text{v},t} = (R_t - V(s_t; \theta_\text{v}))^2 .$$

1.3 Actor Network

Let a_t be the action that was taken at time step t, v_t the value that the value network estimated as the future reward given the current state s_t at time step t (v_t is just an abbreviation for $V(s_t; \theta_\text{v})$) and θ_a the parameters (neural network weights) of the actor network. Furthermore, let β be a factor that specifies the importance of the entropy H. π designates the probability density function, which means that $\pi\,(a_t \mid s_t; \theta_\text{a})$ is the value of the probability density function of taking action a_t in state s_t with the current weights of

Algorithm 1 Partial Action Actor Critic Learning – pseudocode for each learning thread. Only $\theta_{\text{a,g}}$ and $\theta_{\text{v,g}}$ are shared between the threads. They are initialized randomly in the beginning. TODO: Update algorithm

loop
    $l_\text{actions} \leftarrow []$
    $l_\text{states} \leftarrow []$
    $l_\text{values} \leftarrow []$
    $l_\text{rewards} \leftarrow []$
    $l_\text{estimatedValues} \leftarrow []$
    $l_\text{snapshots} \leftarrow []$
    $t \leftarrow 0$
    $s_0 \leftarrow$ initialState()
    $\theta_\text{a} \leftarrow \theta_{\text{a,g}}$
    $\theta_\text{v} \leftarrow \theta_{\text{v,g}}$
    repeat
        l_states.append(s_t)
        $l_\text{estimatedValues}$.append($V(s_t; \theta_\text{v})$)
        if $t \bmod t_\text{max} = 0$ **then**
            $\theta_\text{a} \leftarrow \theta_{\text{a,g}}$
            $\theta_\text{v} \leftarrow \theta_{\text{v,g}}$
            $l_\text{snapshots}$.append(($\theta_\text{a}, \theta_\text{v}$))
        end if
        Sample a_t from the
            Actor Network's probability distribution
        l_actions.append(a_t)
        $l_{\text{partialActions},t} \leftarrow$ partialActions(a_t)
        for all partial actions $a_{\text{p},n,t}$ in $l_{\text{partialActions},t}$ **do**
            Take partial action $a_{\text{p},n,t}$
        end for
        Receive feedback $r_{\text{p},n,t'}$
            where $t' \leq t$ and $0 \leq n \leq \#(l_{\text{partialActions},t'})$
        if all feedback of $a_{t'}$ was received **then**
            $r_{t'} \leftarrow$ reward w.r.t all feedback $r_{\text{p},n,t'}$
            l_rewards.append($r_{t'}$)
            if $\#(l_\text{rewards}) > t_\text{max}$ or the episode is over **then**
                computeGradients
            end if
        end if
        Generate s_{t+1} using $r_{\text{p}_{t'}}$
        $t \leftarrow t + 1$
    until reaching the end the episode
end loop

the actor network θ_a. Then

$$\begin{aligned} l_{\text{a},t} = &- \log\left(\pi\,(a_t \mid s_t; \theta_\text{a})\right) (R_t - v_t) \\ &- \beta H\left(\pi\,(s_t; \theta_a)\right) \end{aligned}$$

is the actor loss that we try to minimize.

2 CONGESTION CONTROL SPECIFICS

The classical Reinforcement Learning assumes that a reward follows an action and vice-versa (see Figure 1). However, in the case of congestion control, it is desirable to perform a new action without having received a reward for the previous action. For example,

Algorithm 2 Partial Action Actor Critic Learning – procedure which computes and applies the gradients.

function COMPUTEGRADIENTS
  $t_{\text{end}} = \min(t_{\text{max}}, \#(l_{\text{rewards}}))$
  $\theta_{\text{backup}} \leftarrow (\theta_{\text{a}}, \theta_{\text{v}})$
  $\theta_{\text{a}}, \theta_{\text{v}} \leftarrow l_{\text{snapshots}}[0]$
  $R_{i+1} \leftarrow \frac{l_{\text{estimatedValues}}[t_{\text{end}}]}{1-\gamma}$
  for $i \leftarrow t_{\text{end}} - 1, 0$ **do**
    $R_i \leftarrow r_i + \gamma R_{i+1}$
    $a \leftarrow l_{\text{actions}}[i]$
    $s \leftarrow l_{\text{states}}[i]$
    $v \leftarrow l_{\text{values}}[i]$
    $d\theta_{\text{v}} \leftarrow \frac{\partial (R_i - v)}{\partial \theta_v}$
    $d\theta_{\text{v}} \leftarrow -\frac{\partial \log(\pi(a \mid s;\theta_{\text{a}}))(R_i - v)}{\partial \theta_a} - \frac{\partial \beta H(\pi(s;\theta_a))}{\partial \theta_a}$
    $\theta_{\text{a,g}} \leftarrow \theta_{\text{a,g}} + d\theta_{\text{a}}$
    $\theta_{\text{v,g}} \leftarrow \theta_{\text{v,g}} + d\theta_{\text{v}}$
  end for
  Remove first t_{end} elements from $l_{\text{actions}}, l_{\text{states}}, l_{\text{values}}, l_{\text{rewards}}, l_{\text{estimatedValues}}$
  Remove first element from $l_{\text{snapshots}}$
  $\theta_{\text{a}}, \theta_{\text{v}} = \theta_{\text{backup}}$
end function

imagine that you receive two acknowledgements directly after each other. For each of these two acknowledgements an action has to be performed but it does not seem feasible for the second action to wait until the first action has received a reward. The Asynchronous Actor Critic framework as described by [1] cannot be applied to congestion control as it assumes that actions and rewards are synchronized and thus we use the proposed Partial Action Actor Critic Learning (see section 1). We describe the concept of using reinforcement learning for congestion control with a petri net (see Figure 3).

In the following a time step t corresponds to the reception of an acknowledgement. The beginning of the flow, before any packet is sent, corresponds to time step 0.

- s_t The state describes the current "congestion state". Various features that indicate congestion are included in it. Each time an acknowledgement is received, the state is updated and the actor network is asked for the next action.
- a_t Based on a given state and the history of previous states, the actor network returns an action a_t, which is a real number (≥ 1) that stands for the congestion window to be used until the next acknowledgement is received.
- r_t The reward is a tuple of at least one reward metric. These are discussed in more detail in subsection 2.1
- v_t The value is a tuple of the expected average reward estimated by the value network (see subsection 2.1) for each of the reward metrics.

We actually use four different types of reward:

- $r_{\text{packet},t}$ is the number of the packets that the sender sent during time step t and that were not lost (so they were acknowledged at some point by the receiver).

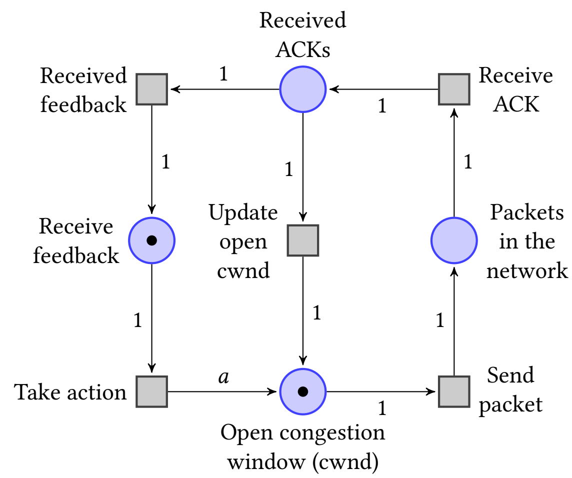

Figure 3: A petri net describing the congestion control mechanism. We start with one token in the state *Open congestion window*, which means that initially we can send one packet. We start with another token in the state *Received feedback*, which means that we can take an action right in the beginning. The concept is the following: If there is at least one token in the *Open congestion window*, send a packet. The acknowledgement (or timeout if the packet gets lost) for the packet is received by the *Open congestion window*, meaning that when we get an ACK we can send another packet (The *Open congestion window* signifies the congestion window minus the packets that are currently unacknowledged.). Furthermore the state *Received feedback* receives each ACK/timeout and takes an action. *Take action* adds *a* tokens to the *Open congestion window*, where a is a real number (possibly also negative); thus in each state there can also be a real number of tokens (e.g. 2.34 tokens are possible in the *Open congestion window*). However, we define that the congestion window can never be smaller than 1.

- $r_{\text{byte},t}$ is the sum of the bytes of the packets that the sender sent and that were not lost. Example: During t, three packets were sent with 300, 200, and 1500 bytes each so $r_{\text{byte},t} = 2000$
- $r_{\text{delay},t}$ is the sum of the round trip times of the packets that the sender sent and that were not lost.
- $r_{\text{duration},t}$ is the sum of the time between receiving the last packet and receiving this packet ("inter-receive time") for the packets that the sender sent and that were not lost.

2.1 Value Network

In the case of congestion control, the loss function $l_{\text{v},t}$ of the value network is actually the sum of the square of the difference for each of the four moving averages for each type of reward:

$$l_{v,t} = \left(R_{\text{packet},t} - V_{\text{packet}}(s_t;\theta_v)\right)^2 + \left(R_{\text{byte},t} - V_{\text{byte}}(s_t;\theta_v)\right)^2 + \left(R_{\text{delay},t} - V_{\text{delay}}(s_t;\theta_v)\right)^2 + \left(R_{\text{duration},t} - V_{\text{duration}}(s_t;\theta_v)\right)^2$$

2.2 Actor Network

The actor network outputs two parameters: The mean of a normal distribution μ and its standard deviation σ.

Each time an action a_t is requested, a value X is sampled from the current normal distribution defined by the parameters μ and σ:

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

a_t is the sampled value X at time t and the window is incremented by a_t; however, the window can never be smaller than 1. At the beginning of a flow the window is 1 as well.

With H being the entropy, the actor network minimizes the loss

$$l_{a,t} = -\log\left(\pi\left(a_t \mid s_t;\theta_a\right)\right) \left(\left(\log\left(\frac{R_{\text{byte},t}}{R_{\text{duration},t}}\right) - \log\left(\frac{v_{\text{byte},t}}{v_{\text{duration},t}}\right)\right) - \left(\frac{R_{\text{delay},t}}{R_{\text{packet},t}} - \frac{v_{\text{delay},t}}{v_{\text{packet},t}}\right)\right) + \beta H\left(\pi\left(s_t;\theta_a\right)\right).$$

2.3 Example

(1) Receive an ACK
(2) Update the internal state to reflect the information that this ACK provides (for example, the state consists of the round trip time experienced by this packet, the time since we receive the last ACK etc.)
(3) Sample an increase for the congestion window from the Actor Network (e.g. 0.24) and add it to the current congestion window.
(4)(a) If this was the last ACK, which completes a previous action (e.g. a previous action resulted in 3 packets being sent; so if this is the ACK for the third (and last) packet of this previous action we got all the ACKs to compute the reward.)
   (b) If we now have t_{max} rewards (set to 20) then we also update the neural network.

2.4 Convergence

Assume we have N senders, each with its specific baseline delay (this encompasses all delays excluding only the queueing delay) d_i. Each sender keeps a congestion window b_i. The link speed at the bottleneck is designated as c.

With $b = (b_i)$ we observe the following relation:

$$c = \sum_{j=1}^{N} \frac{b_j}{d_j + d_q(b)}$$

Then we define the utility function for each sender i as

$$V_i(b) := \begin{cases} \log\left(\frac{b_i}{d_i}\right) - \delta_i d_i & \text{if } \sum_{j=1}^{N} \frac{b_i}{d_i} \le c \\ \log\left(\frac{b_i}{d_i+d_q(b)}\right) - \delta_i(d_i + d_q(b)) & \text{otherwise} \end{cases}$$

$$\log\left(\frac{b_i}{d_i + d_q}\right) - \delta_i(d_i + d_q) = \log(b_i) - \log\left(d_i + d_q\right) - \delta_i(d_i + d_q)$$

$$\frac{\partial V_i}{\partial b_i} = \begin{cases} \frac{1}{b_i} & \text{if } \sum_{j=1}^{N} \frac{b_i}{d_i} \le c \\ \frac{1}{b_i} - \frac{\partial d_q}{\partial b_i}(b_i)\frac{1}{d_i+d_q(b_i)} - \delta\frac{\partial d_q}{b_i}(b_i) & \text{otherwise} \end{cases}$$

To show diagonal strict concavity we have to show that d.s.c. is fulfilled (1) for two distinct points lying in $\sum_{j=1}^{N} \frac{b_i}{d_i} \le c$ (2) for two distinct points lying in $\sum_{j=1}^{N} \frac{b_i}{d_i} > c$ and (3) when one point is in one part of the function and the other in the other one.

TODO: Insert what d.s.c actually is and quote Altman's paper

3 RESULTS

We train on the following scenario:

Parameter	Value	Distribution
Two-way propagation delay	100 ms	constant
Bottleneck bandwidth	3 Mbit/s	constant
Number of senders	2	constant
Flow length	∞	constant
Simulation duration	50 s–150 s	uniform
Buffer size	∞	constant
Stochastic loss prob.	0%	constant

Table 1: The parameters of our very first simple experiment

TODO: Insert results when comparing to regular TCP

REFERENCES

[1] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.