Congestion Control

October 10, 2017

Congestion Control

- Why Congestion Control? Before congestion control was invented: Everyone sent as much as they pleased $\rightarrow$ *Congestion Collapse*.
- Goal: Estimate available bandwidth. Don't send too much, don't send too little.
- Method: Keep a *Congestion Window*
- E.g. Congestion Window of 5 means that we can have up to 5 packets somewhere in the network.

How does Congestion Control work nowadays?

Simplified:

- Start with Congestion Window being 1 in the beginning of a flow.
- Upon receiving an acknowledgement for a previously sent packet, increase the Congestion Window (cwnd):

$$\text{cwnd} = \frac{1}{\text{cwnd}}$$

- Recently also a bit more sophisticated $\rightarrow$ *CUBIC* etc.
- When there is packet loss then we sent too much (buffer of a router on the way overflew) $\rightarrow$ congestion $\rightarrow$ decrease Congestion Window
- Currently the most common thing to do is

$$\text{cwnd} = \frac{\text{cwnd}}{2}$$

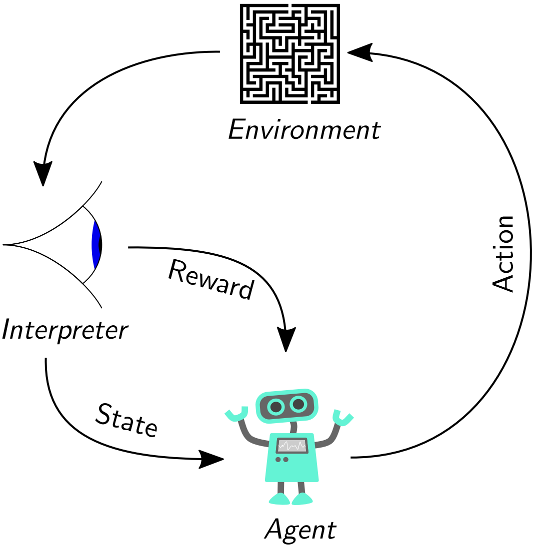
Problems

- Only decrease window on loss $\rightarrow$ then it's already too late! We should decrease before when buffers of routers fill up and latency increases!
- On wireless connections stochastic packet loss is common $\rightarrow$ TCP thinks it's congestion.

Potential Solution

- Let's build some machine learning thing!
- Solutions already exist $\rightarrow$ *TCP ex Machina* by Winstein and Balakrishnan (2013).
- They simulate networks and learn an optimum congestion control more or less by using a brute force algorithm.
- Example: Use networks with 1 to 5 senders, RTT from 10 to 100 ms. Use one set of congestion control rules for 1000 simulations. Then change some parameters and check if it improved (actually they do it in a smarter way)
- Problem: Training has to be done offline. But it would be nice to have a Congestion Control that learns in real time, online!

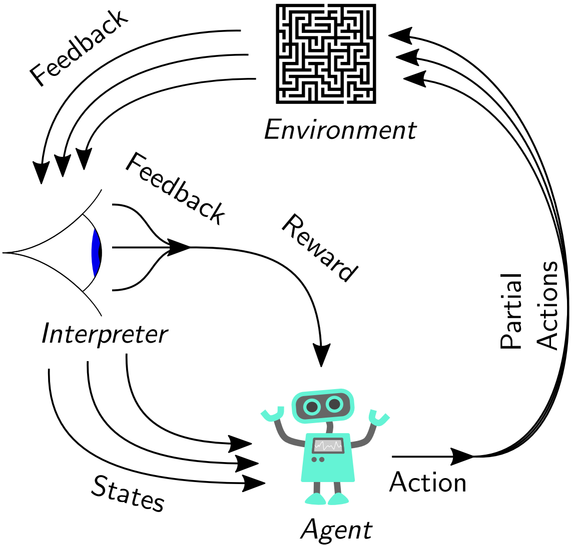
Reinforcement Learning



Problems with RL

- Let's say an action is increasing the Congestion Window
- We can calculate the reward when we get back all the ACKs of packets that were sent during an action
- This means we can't do anything until receiving each action's reward. But that takes at least one Round Trip Time! Doesn't work.
- Example: We increase the cwnd by 2 packets. So we can send two packets. To evaluate if this action was good we have to get the ACKs of these two packets, which happens after one RTT! In the meantime another ACK could have arrived. What do we do? Not defined with RL...

Partial Actions



Partial Actions: Example

- We increase the window by 0.3 (action). This allows us to send two packets (partial actions).
- We receive the ACK for the first packet (feedback). We update the state and perform a new action.
- We receive the second ACK. Again, we update the state and perform an action. However, because we got all partial rewards of the previous action, we can calculate the reward and update our agent.

Key point: We can update the state without receiving the full reward yet.

Asynchronous Actor Critic

- A Deep Learning framework for reinforcement learning. Used for learning how to play video games.
- Maximize long term reward:

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

- The **Critic** tries to estimate how much (long-term) reward he can expect considering the current state.

Asynchronous Actor Critic – Actor

- The **Actor** tries to perform an action that is better than what the Critic would expect.
- It outputs two things:
  - What it thinks is the best action (e.g. increase the window by 0.3)
  - A standard deviation to experiment a little bit (e.g. 0.45)
- So the actor outputs a distribution from which actions are sampled. Thanks to the standard deviation we experiment and don't get stuck with suboptimal actions.

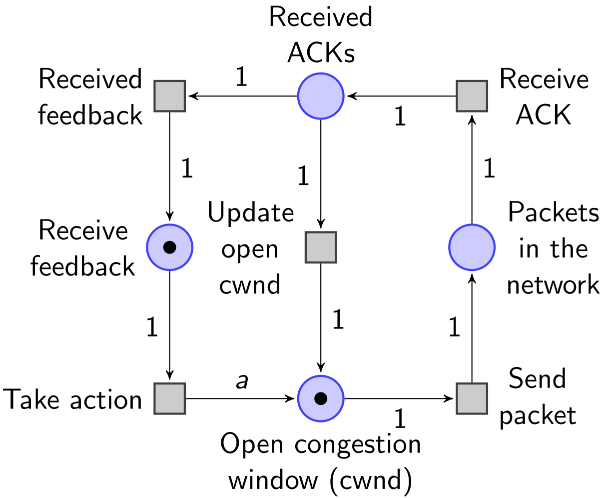
Preliminary results – Experiment characteristics

Parameter	Value	Distribution
Two-way propagation delay	150 ms	constant
Bottleneck bandwidth	15 Mbit/s	constant
Number of senders	8	constant
Flow length	100 kB	exponential
Time between flows	0.5 s	exponential
Simulation duration	100 s	constant
Buffer size	1000 packets	constant
Stochastic loss prob.	0%	constant

Preliminary results – Comparison

Parameter	Value	Distribution
Two-way propagation delay	150 ms	constant
Bottleneck bandwidth	15 Mbit/s	constant
Number of senders	8	constant
Flow length	100 kB	exponential
Time between flows	0.5 s	exponential
Simulation duration	100 s	constant
Buffer size	1000 packets	constant
Stochastic loss prob.	0%	constant

Petri net


Received
ACKs
Received
feedback
Receive
ACK
1
1
1
1
1
Update
open
cwnd
Receive
feedback
Packets
in the
network
1
1
1
Take action
a
Send
packet
1
Open congestion
window (cwnd)