A Deep Reinforcement Learning Perspective on Internet Congestion Control

by Nathan Jay*, Noga H. Rotman*, Brighten Godfrey, Michael Schapira, and Aviv Tamar

*Equal contribution
Internet Congestion Control

The Internet
(maybe?)

End Host

Server
Internet Congestion Control

The Internet
(maybe?)

End Host

Data

t=1

Server
Internet Congestion Control

The Internet

End Host

Server

(maybe?)
Internet Congestion Control

The Internet

End Host

Server

Data

$ t = 5.1 $
Internet Congestion Control

The Internet
(maybe?)

End Host

Server

Ack
t=5.2
Internet Congestion Control

End Host

The Internet
(maybe?)

Server
Internet Congestion Control

The Internet
(maybe?)

End Host

Server

Ack

t=10.2
Internet Congestion Control

The Internet (maybe?)

End Host

Data

Server

Data

Ack

Ack

Data

t=1

t=10.2

Data

t=5.1

Data

t=5.2
Internet Congestion Control

Latency Trace of Internet Path*

*from pantheon.stanford.edu
Internet Congestion Control

Latency Trace of Internet Path*

*from pantheon.stanford.edu
Internet Congestion Control

Latency Trace of Internet Path*

*from pantheon.stanford.edu
Internet Congestion Control

Latency Trace of Internet Path*

Underlying Complexity:

- Enormous, dynamic network
- Massive agent churn
  ~80,000 agents/second
- Very little information

*from pantheon.stanford.edu
Revisiting Congestion Control

Congestion Control Timeline

- 1988
- Flavors of TCP Congestion Control
  (Tahoe, Reno, Cubic, Illinois, Vegas, ...)
- 2016
- 2019

- Same network model
- Same action space
- Slightly different control algorithms
Revisiting Congestion Control

Congestion Control Timeline

1988 - Flavors of TCP Congestion Control
(Tahoe, Reno, Cubic, Illinois, Vegas, ...)

- Same network model
- Same action space
- Slightly different control algorithms

2016 - Introduction of QUIC, replaces significant amount of Google traffic.

- New models
- New action space (packet pacing added to Linux)
- Novel control algorithms and research (BBR, Copa, PCC)
Reward-based architecture: PCC

Observations

Test Rates

Network

Performance Statistics

Monitor Interval

Input Features:
1. Send Ratio
2. Lat. Ratio
3. Lat. Inflation
Reward-based architecture: PCC

Observations

Network

Test Rates

Performance Statistics

Monitor Interval

Input Features:
1. Send Ratio
2. Lat. Ratio
3. Lat. Inflation

Actions

Utility vs Rate

Link Capacity
Agent Architecture

Input Features:
1. Send Ratio
2. Lat. Ratio
3. Lat. Inflation

Rate Change Factor

\[
\alpha > 0: \text{Old Rate} \times (1 + w\alpha)
\]

\[
\alpha < 0: \frac{\text{Old Rate}}{1 - w\alpha}
\]
Agent Architecture

Key Design Choice: Scale-free observations affect robustness

Rate Change Factor

\[
\text{New Rate} = \begin{cases} 
\alpha > 0: \text{Old Rate} \times (1 + w\alpha) \\
\alpha < 0: \text{Old Rate} / (1 - w\alpha)
\end{cases}
\]
Training Environment:

- Simulated network
- Each episode chooses link parameters from a range

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Latency</th>
<th>Loss</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 6mbps</td>
<td>50 - 500ms</td>
<td>0 - 5%</td>
<td>1 - ~3000pkt</td>
</tr>
</tbody>
</table>

- **Standard gym** at [github.com/PCCProject/PCC-RL](http://github.com/PCCProject/PCC-RL)
Training/Testing Environment

Training Environment:

- Simulated network
- Each episode chooses link parameters from a range:

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Latency</th>
<th>Loss</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 6mbps</td>
<td>50 - 500ms</td>
<td>0 - 5%</td>
<td>1 - ~3000 pkt</td>
</tr>
</tbody>
</table>

- **Standard gym** at github.com/PCCProject/PCC-RL

Testing Environment:

- Real packets in Linux kernel network emulation
- Much wider testing range:

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Latency</th>
<th>Loss</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 128mbps</td>
<td>1 - 512ms</td>
<td>0 - 20%</td>
<td>1 - 10000 pkt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Latency</th>
<th>Loss</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 128mbps</td>
<td>1 - 512ms</td>
<td>0 - 20%</td>
<td>1 - 10000 pkt</td>
</tr>
</tbody>
</table>
State-of-the-art Results

Test Description:

- Emulated network, with real Linux kernel noise
- Time-varying link
State-of-the-art Results

Test Description:

- Emulated network, with real Linux kernel noise
- Time-varying link

![Emulated Dynamic Link Performance](chart.png)
State-of-the-art Results

Test Description:

- Emulated network, with real Linux kernel noise
- Time-varying link

![Diagram showing Emulated Dynamic Link Performance]
State-of-the-art Results

Test Description:

- Emulated network, with real Linux kernel noise
- Time-varying link
State-of-the-art Results

Test Description:

- Emulated network, with real Linux kernel noise
- Time-varying link

Aurora is on the Pareto front of state-of-the-art algorithms
Exciting Directions

- Multi-agent scenarios:
  - Cooperative
  - Selfish

- Online training:
  - Few-shot training
  - Meta-learning

- Multi-objective Learning:
  - File transfer
  - Live video

By The Opte Project - Originally from the English Wikipedia; description page is/was here., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=1538544
See us at:

Poster #45
6:30pm - 9:00pm
Pacific Ballroom

Code available at
github.com/PCCProject/PCC-RL