TECHNISCHE UNIVERSITÄT WIEN

# LFQ: Online Learning of Per-flow Queuing Policies using Deep Reinforcement Learning

Maximilian Bachl, Joachim Fabini, Tanja Zseby

Technische Universität Wien, Vienna, Austria

# "Bufferbloat"

- If queues in routers/switches too small: **Underutilization**
- **Solution:** Make queues very **large** for maximum throughput!
- **New problem:** Packets **wait** a long time (several seconds) in the queue: Bufferbloat

# CoDel
## Active Queue Management against Bufferbloat

- Moving time window of 100 ms
- Queuing Delay **must be** $<$ **5 ms** once in each window
- Otherwise: Drop packet(s)

# Fair Queuing

- Separate each flow in separate queue
- No flow can "steal" bandwidth
- Popular implementation for Linux: `fq`

# Fair Queuing

- Separate each flow in separate queue
- No flow can “steal” bandwidth
- Popular implementation for Linux: `fq`

**Static buffer size:**

Buffer often too large or too small

## `fq_codel` Combining fair queuing with CoDel

- State-of-the-art
- Separate queues
- Each queue managed by CoDel
- Keeps each flow's queue $< 5\,\mathrm{ms}$
- Linux implementation: `fq_codel`

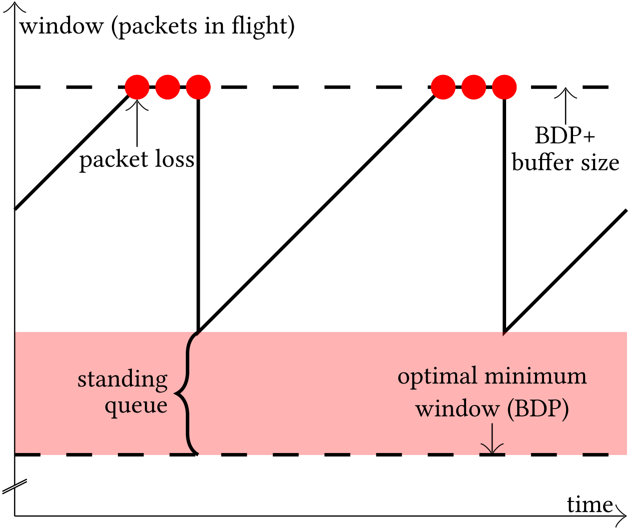
## `fq_codel` **Combining fair queuing with CoDel**

- State-of-the-art
- Separate queues
- Each queue managed by CoDel
- Keeps each flow's queue $< 5\,\mathrm{ms}$
- Linux implementation: `fq_codel`

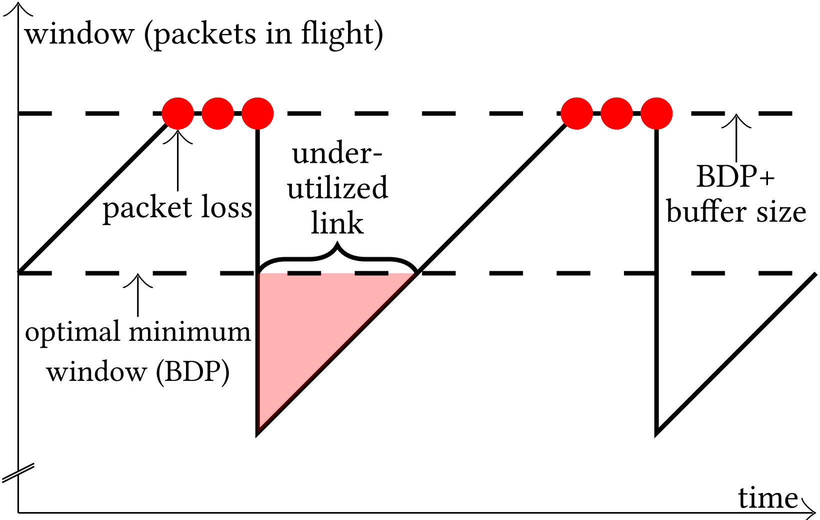
**Inadequate interaction with Congestion Control:**

*Cubic* doesn't achieve full throughput or keeps standing queue

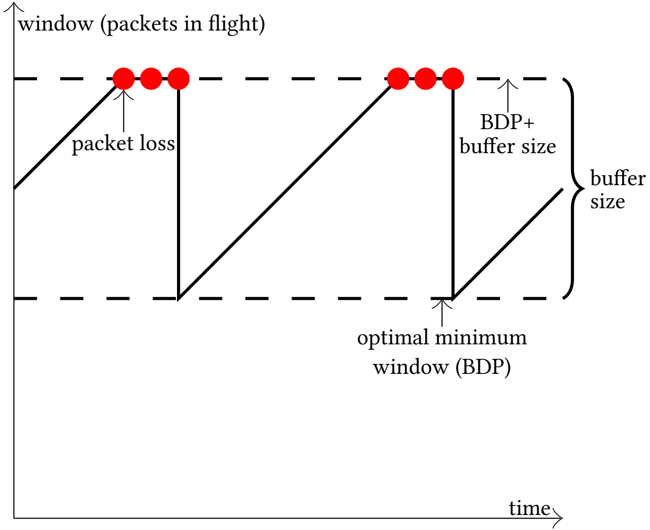
# Buffer too large



# Buffer too small



# Buffer perfect



# cocoa qdisc[1]

- Fair queuing-based qdisc which **adapts** buffer **depending on congestion control**
- Manages to achieve optimal throughput while keeping **delay** from buffering **minimal**
- **Works** for common congestion controls

[1] *Cocoa: Congestion Control Aware Queuing*, Bachl, Fabini, Zseby, 2020

# cocoa qdisc[1]

- Fair queuing-based qdisc which **adapts** buffer **depending on congestion control**
- Manages to achieve optimal throughput while keeping **delay** from buffering **minimal**
- **Works** for common congestion controls

**Hand-crafted algorithm:**

Works well for current congestion controls but might fail badly for new congestion controls.

[1] *Cocoa: Congestion Control Aware Queuing*, Bachl, Fabini, Zseby, 2020

# Deep Learning for Buffering

- Have a separate queue for each flow in a switch/router (fair queuing)
- Define some reward function (like high throughput, low delay)
- Use **Deep Learning**
  - to **fingerprint** each flow
  - and to use the **correct buffer size** for the flow **based on experience**, which **maximizes the reward**

# Reward

$$\mathit{Reward} = \mathit{bandwidth} - \alpha \times \mathit{queue\ size}$$

# Input features

- queue size
- standard deviation of the queue size
- maximimum allowed buffer size
- rate of incoming data
- rate of outgoing data
- time since the last packet loss

# Input features 2

- Don't use these features themselves but exponentially weighted moving averages
- Advantage: No need to keep data around except for features themselves

# Neural Network

- It is a regression problem: Predict optimal buffer size for the flow based on the inputs
- Fully connected neural network with three layers and leaky ReLU
- Outputs optimal maximum buffer size each time packet is received
  - To save compute it is possible, for example, only output optimum buffer size every 100 ms

# Implementation

- ns-3 for network simulation
- Pytorch's C++ API for Deep Learning
- Integrate Pytorch into ns-3
- Develop queuing discipline based on fair queuing
- Each flow is managed by Reinforcement Learning based on Deep Learning

# Offline Learning

- Randomly sample bandwidth, delay, congestion control and flow duration
- For each flow, at random time, launch experiment
- Experiment is A/B testing:
  - Continue with current buffer size +1 and current buffer size -1 (two different experiment)
  - Let the neural network learn buffer size that performed better regarding reward
- Train using L1 loss (mean absolute error), using a couple of results together as a batch

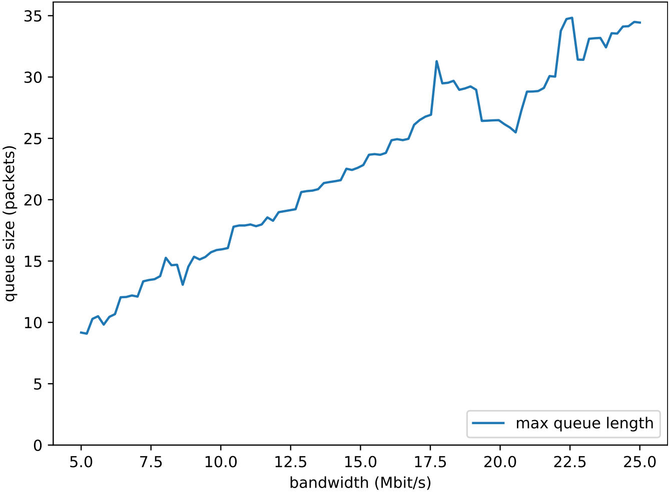
Training takes several hundred thousand flows to converge.

# Visualizing success

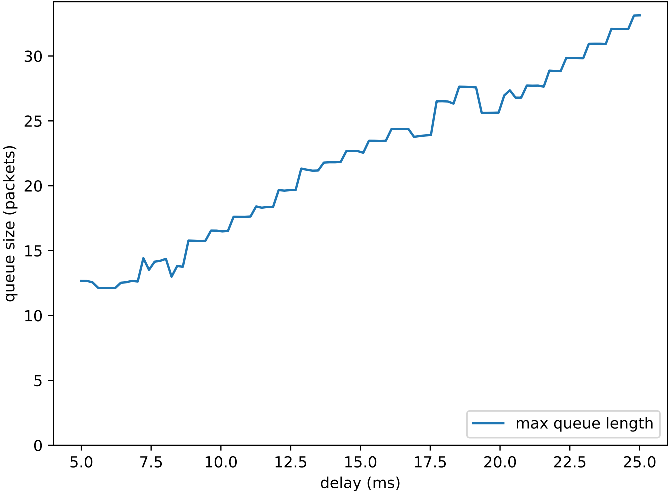
Should learn

- Larger bandwidth $\rightarrow$ larger buffer
- Larger delay $\rightarrow$ larger buffer

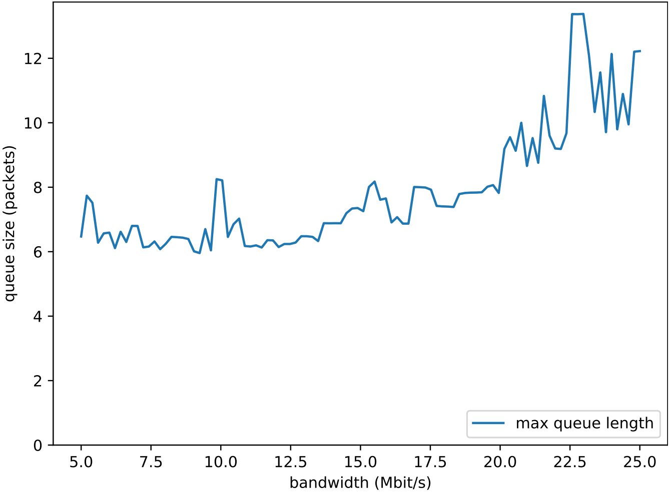
# New Reno, varying bandwidth



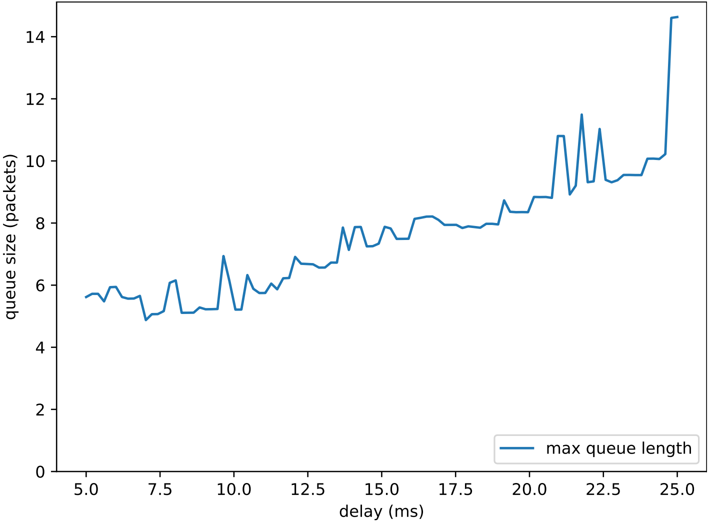
# New Reno, varying delay



# Bic, varying bandwidth



# Bic, varying delay



# Correlations

Correlations calculated of each of the above plots:
Correlation of 100% $\rightarrow$ Our Reinforcement Learning system learned successfully

Table: Correlation between bandwidth/delay of the link and output buffer size.

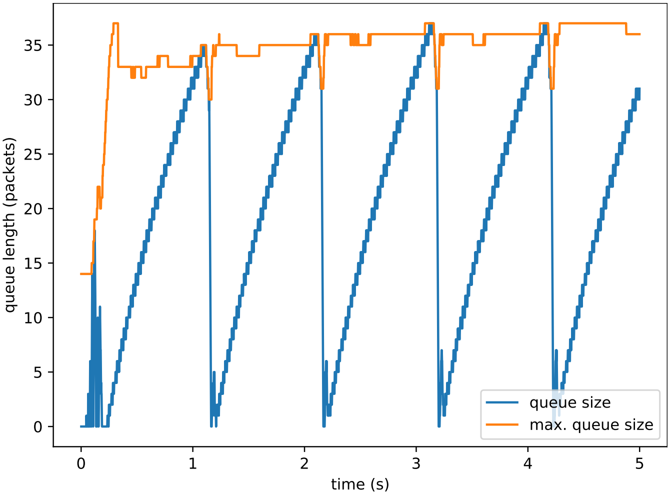
| Congestion control | bandwidth | delay |
|---|---|---|
| New Reno | 98.3% | 99.1% |
| BIC | 80.2% | 90.2% |

# Average queue sizes

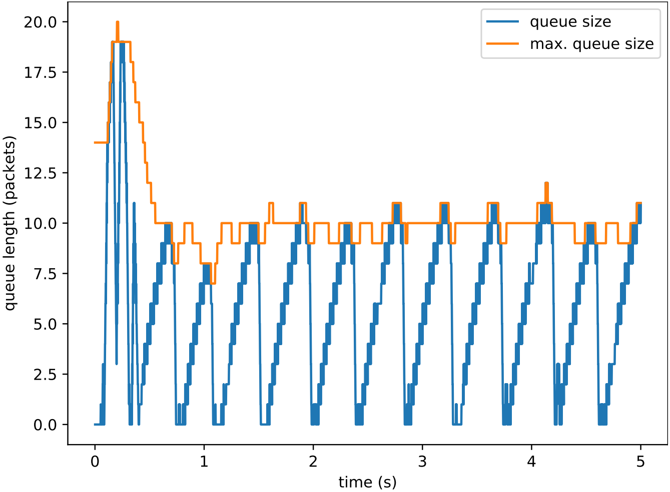
Average queue is smaller for New Reno: This is expected since it has a larger multiplicative decrease parameter!
$\rightarrow$ New Reno needs a larger buffer on average so that the buffer never becomes empty. Our mechanism learned that!

| Congestion control | avg. max. queue length |
|---|---:|
| New Reno | 22 |
| BIC | 8 |

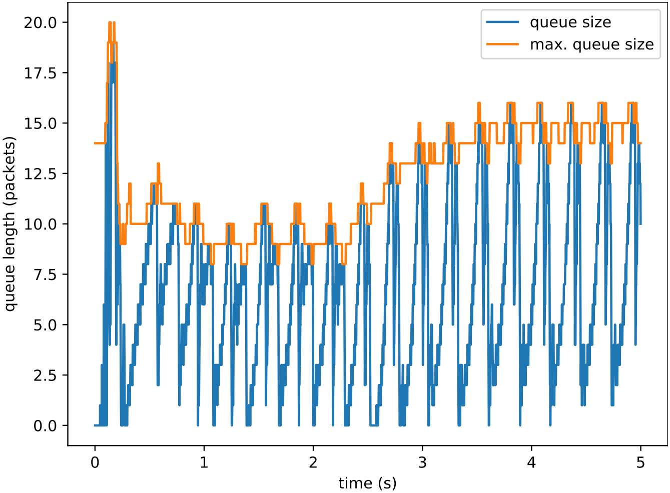
# 25 Mbit/s, 15 ms, New Reno, LFQ



# 6 Mbit/s, 15 ms, New Reno, LFQ



# 25 Mbit/s, 15 ms, BIC, LFQ



# Correlations

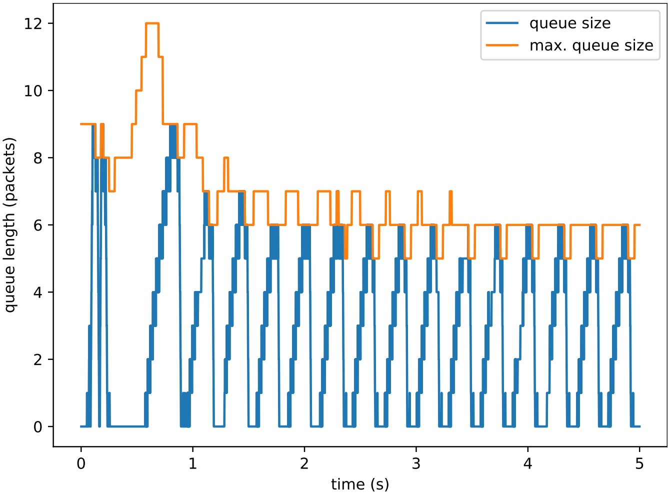
Table: Correlation between bandwidth/delay of the link and output buffer size.

| Congestion control | bandwidth | delay |
|---|---|---|
| New Reno | 99.3% | 96.5% |
| BIC | 75.4% | 62% |

## Average queue sizes

| Congestion control | avg. max. queue length |
|---|---:|
| New Reno | 13 |
| BIC | 5 |

# 6 Mbit/s, 15 ms, New Reno, LFQ



# Online Learning

Same as Offline Learning except:

- Don't do A/B testing
- Instead: Use second neural network that outputs expected reward (*Critic Network*) (trained using L2 loss (mean squared error).
- Either perform experiment A **or** experiment B (either current buffer size -1 packet or +1 packet)
- If result was better than what the Critic Network expected, let first neural network (*Actor Network*) learn to output this buffer size in the future.

Training takes longer (several million training flows)

# Correlations

Table: Correlation between bandwidth/delay of the link and output buffer size.

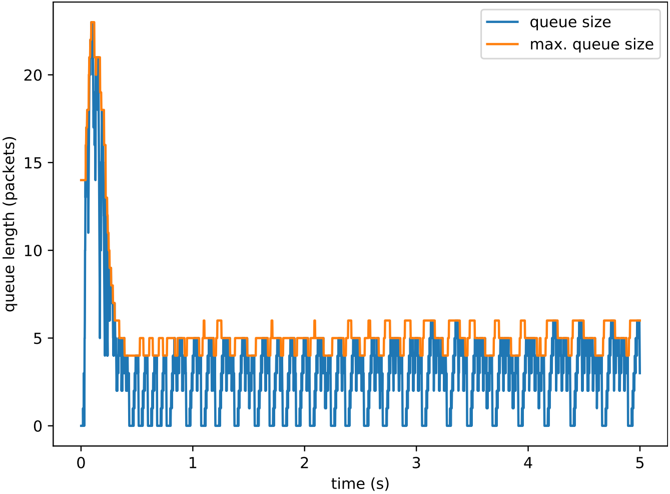
| Congestion control | bandwidth | delay |
|---|---|---|
| New Reno | 86.1% | 87.7% |
| BIC | 72.9% | 73.6% |

# Average queue sizes

| Congestion control | avg. max. queue length |
|---|---:|
| New Reno | 14 |
| BIC | 8 |

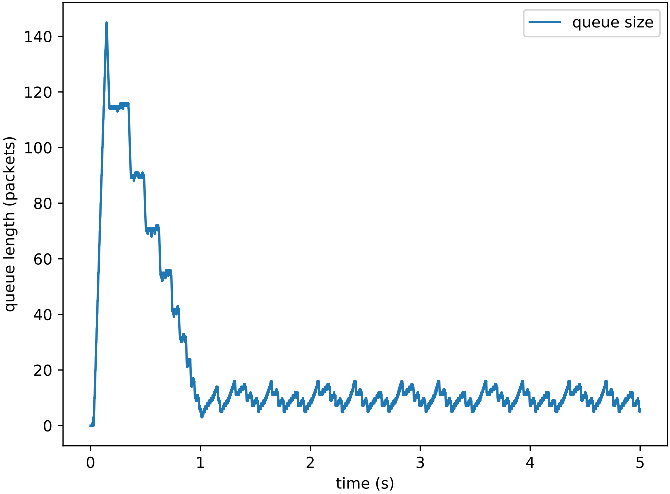
# Example: Our solution

## 15 Mbit/s bandwidth, 5 ms delay



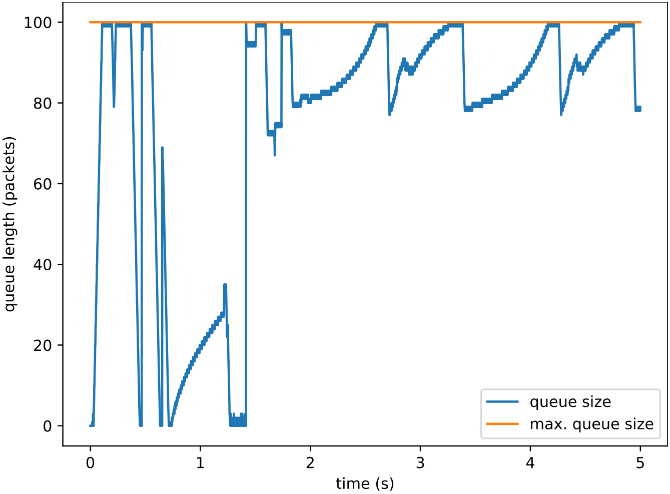
# Example: FqCoDel (default on home routers)

## 15 Mbit/s bandwidth, 5 ms delay



# Example: Fifo (default on Linux)

## 15 Mbit/s bandwidth, 5 ms delay



# Systematic comparison

Table: 400 experiments; delay 5 – 25 ms; bandwidth 5 – 25 Mbit/s; New Reno and Bic Congestion Control. Results averaged.

| | avg. throughp. | queue size max. | queue size avg. |
|---|---|---|---|
| LFQ, offline $\alpha = 0.01$ | 13.4 | 23.9 | 7.7 |
| LFQ, offline $\alpha = 10$ | 12.5 | 12.7 | 3.4 |
| LFQ, online $\alpha = 10$ | 12.8 | 16.1 | 4.5 |
| FqCoDel | 13.7 | 155.4 | 15.4 |
| fq 100 | 11.7 | 100 | 51.1 |
| fq 1000 | 11.9 | 1000 | 630.4 |

# Observations

- Scaling of features very important
- Pytorch better to integrate with other code (ns-3) than TensorFlow
- We thought simulating a flow in ns-3 would be faster than running it in the real world. It is not.

# Conclusion

- LFQ is based on fair queuing
- It fingerprints each flow
- It learns to optimize a reward function
- It achieves high throughput and low delay when compared to competing solutions
- It has low computational overhead
- We envision deployment close to end users (on switches, routers)

# LFQ: Online Learning of Per-flow Queuing Policies using Deep Reinforcement Learning

Maximilian Bachl, maximilian.bachl@tuwien.ac.at
Joachim Fabini
Tanja Zseby

Technische Universität Wien, Vienna, Austria