Xavier: A Reinforcement-Learning Approach to TCP Congestion Control

CS221 Final Report

Akshay Agrawal
akshayka@stanford.edu
July 24, 2016

In this paper, I present Xavier, a congestion control policy informed by reinforcement learning and a first step towards adaptible control and analyze its performance on two simulated network topologies. Experimental results hint at the utility of using reinforcement-learning for congestion control while also revealing difficulties entailed by doing so.

1 Introduction

Controlling congestion is a fundmanetal problem in computer networks. If the input load is greater than the output bandwidth at a particular switch, the bottleneck's queue begins to fill up and we say that it is congested. In pathological scenarios and under certain protocols, the saturation of buffers, or bufferbloat [5], can lead to congestion collapse, a condition in which congestion reaches a level sufficient to limit useful communication and encourage redundant packet retransmissions [9]. The twin problems of bufferbloat and congestion control have come into sharp relief over the past decade. In 2002, 100 GB of IP traffic were trasmitted globally per second; in 2014, that number rose to 16,144 GB/s, and by 2019, Cisco predits that it will rise to 51,794 GB/s [7].

Congestion control is fundamentally a Markovian process — that of deciding whether or not to send a packet at a given time step [13]. Protocol designers have hand-rolled policies for TCP and all its flavors; these policies maintain congestion control windows, or bounds on the number of outstanding (unacknowledged) bytes that the sender will tolerate at any given moment. The policies vary in how they manage the window, but what is common among them is that they are heuristically-driven and designed without explicit objective functions in mind [13]. Indeed, computer scientists (and economists) have struggled to characterize the teleology of TCP. As such, it is not surprising that,

when compared to Remy, a congestion avoidance algorithm that was generated by explicitly modeling TCP as an end-to-end, cooperative, non-zero sum and Markovian game, congestion control algorithms like TCP Tahoe, Reno, New Vegas, and even the in-network XCP and Cubic-over-sfqCoDel perform poorly, where performance is measured by average throughput and delay for senders in a particular experiment [13].

While Remy does outperform existing TCP congestion control algorithms, it resembles them insofar as it is an *inflexible* algorithm. Remy is designed offline, where it learns a policy via repeated simulation trials; however, once deployed, Remy's mappings from states to actions remains fixed. Because Remy is designed for specific network topologies and loads, its performance falls below traditional congestion control algorithms as soon as the network conditions deviate from those for which it was designed [13]. And so we are left with a less-than-ideal trade-off: Either we settle for suboptimal performance that generalizes, or strive for better performance and lock ourselves into fixed network conditions in the process.

In this paper, I explore the question: Is it possible to circumvent the trade-off between generalizability and throughput-delay by designing congestion control algorithms that learn online? In particular, I investigate whether TCP-compatible congestion control algorithms informed by Q-learning might 1) acheive high throughput and low delay and 2) generalize across varying network topologies.

2 Task Definition

We observe that the problem of congestion control can be modeled as a Markovian process — at each time step, the controller must decide whether or not to send a packet; equivalently, the controller must determine the packet send rate. The state needed to inform controller actions does not depend historical states or actions, as we will see in section IV. The controller has two (competing) goals: maximizing throughput and minimizing delay.

To make this definition more concrete, in this paper, all of the discussed algorithms modulate the packet send rate by maintaining a *congestion control window* (CWND), which determines the amount of bytes the sender can ship over the network when it decides to do so. Packet send events are triggered by the receipt of acknowledgements (ACKs); in other words, we discretise the problem of modulating the packet send rate on a per-ack basis. In particular, upon the arrival of an ACK, the controller performs an action upon the CWND. This controller definition is in line with that adopted by the majority of popular TCP congestion control algorithms [9], and by Remy as well [13].

3 Literature Review

A sizeable amount of work has gone into building smarter congestion control algorithms. Winstein, et. al push TCP to its limits by performing an exhaustive search for an optimal congestion control policy for a given network, modeling the problem as a partially observable decentralized decision process. These policies, however, fail to generalize beyond their trained domains [13]. Dong, et. designed PCC, a policy that learns online by measuring the impact controller actions have upon performance and performing gradient ascent. While PCC's performance is promising, widespread deployment seems unlikely as it cannot be easily integrated into existing TCP implementations, unlike Remy [4].

The authors in [1] apply Q-learning to packet routing, as do those in [2], [10], and [3]. [6] and [11] both apply reinforcement learning to congestion control, though their algorithms are designed for specialty networks (multimedia and ATM, respectively).

4 Models

In this section, I present three classes of models: baselines, oracles, and Xavier. The baselines greedily maximize either throughput or minimize delay; the oracles are all variants of Remy [13]; and Xavier is the algorithm that is the namesake of this paper.

4.1 Greedy Maximizers as Baselines

For a given set of network conditions, I define my oracle as the Remy congestion control algorithm that was explicitly designed for those conditions.

I define two baselines: one that greedily maximizes throughput and one that greedily minimizes delay. The former algorithm simply keeps its congestion control window at 40 times the maximum segment size (MSS). The latter sends packets one-by-one, keeping the window at 1 MSS. Intuitively, if the throughput maximizer acheives low delay in a given network, then it must be the case that the experimental conditions did not necesitate congestion control. And if the delay minimizer itself leads to high delays, then it is likely that, no matter what controller is deployed, the network will experience congestion. Thus, these baselines allow us to determine whether the problem of congestion control is relevant and tractable one for particular networks.

4.2 Remy as an Oracle

For a given set of network conditions, I define my oracle as the Remy congestion control algorithm that was explicitly designed for those conditions. Remy models are created by exhaustively searching for an optimal control policy for a given network, and I, as others have done [4], treat their performance as an approximation for points on the Pareto-optimal tradeoff curve between throughput and delay.

4.3 Xavier: A Reinforcement Learning Congestion Control Algorithm

Xavier is a Q-learning informed congestion control algorithm. In particular, I frame the problem of maintaining a sending packets as a Markov Decision Problem and use SARSA with linear function approximation to learn Q-values.

4.3.1 Action Space

Like TCP, Xavier approaches congestion control by maintaining a CWND; for each ACK it receives, Xavier picks among four actions:

1. exponential growth: $CWND := CWND + 1$
2. linear growth: $CWND := CWND + \frac{1}{CWND}$
3. linear decrease: $CWND := CWND + \frac{1}{CWND}$
4. no-op: $CWND := CWND$

Actions are triggered by the receipt of non-duplicate ACKs — for each ACK received, the controller chooses and executes and action. An epsilon-greedy approach was used, $\epsilon = 0.2$, in order to encourage the controller to explore the state space. In computing q-values, Xavier uses a linear function approximation.

4.3.2 State Space

The state space is informed by two key variables, EWMA-RTT and RTT-RATIO. The former is an exponentially weighted moving average of the round trip time — upon arrival of a new ACK, EWMA-RTT is updated to be 0.8 times its current value plus 0.2 times that of the RTT that arrived. While EWMA-RTT serves as a proxy for delay, RTT-RATIO serves as a proxy for congestion — it is defined as the most recently seen RTT divided by the smallest RTT ever observed. The intuition is that the smallest RTT ever observed likely corresponds closely to the RTT expected if the network switch queues were empty. If RTT-RATIO is one, then it will be likely that packet queues are empty or

close to empty. Ratios are stored with the precision of one decimal point, to avoid overfitting to small changes in the ratio.

Each state is decomposed into the following identity feature templates:

1. RTT-RATIO-Equals ____
2. EWMA-RTT-Equals ____
3. ACTION-Equals ____

The state additionally contains the *cross-features* 1x3 and 2x3 – that is, features are generated for (RTT-RATIO, ACTION) pairs and (EWMA-RTT, ACTION) pairs. The reason for doing so is to capture the local effects that actions have upon particular ratios and expected RTTs.

4.3.3 Reward Function

Each state, action pair triggers k rewards, where k is the number of packets sent as a result of the update to CWND. In particular, the i-th reward, i from 1 to k, is defined as the percent change Δ in the EWMA-RTT, where the current EWMA-RTT is compared against that measured at the time of the control action responsible for generating packet i (call this EWMA-RTT-PREV, plus a positive offet β if the action grew the congestion control window:

$$\Delta = \frac{\text{EWMA-RTT-PREV} - \text{EWMA-RTT}}{\text{EWMA-RTT-PREV}}$$

$$r = \Delta + \beta$$

Observe that Δ is positive when the expected RTT decreases, zero when it does not change, and negative when it increases. The parameter β serves to pressure the controller to send packets.

5 Experimental Design

I am using Network Simulator [8], version 2 — also known as ns-2 — to both design algorithms and run experiments. In particular, I am building off the custom flavor of ns-2 designed by the authors in [13]; doing so allows me to contextualize my results, as I can directly compare the performance of my algorithms with others'. Another argument for using ns-2 is reproducibility: ns-2 is the perhaps the most-used network simulation software suite. An obvious drawback, however, is that of fidelity to the real world, or the possible lack thereof. Unfortunately, I do not have the resources to run experiments on actual test beds.

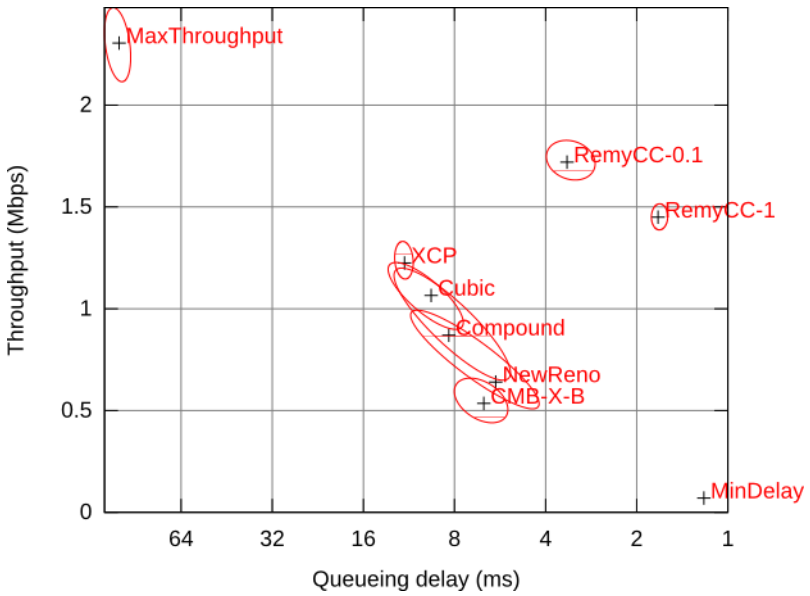

Figure 1: Figure 1. Dumbbell, 15MB Bottleneck. Remy models were trained for this topology, and appear to form a Pareto-optimal frontier.

Experiments were run on a dumbbell topology that has eight senders, eight receivers, and a single bottleneck link in between them. Each sender is paired uniquely to a receiver. Senders switch between on and off, as drawn from an exponential distribution; while on, packets are drawn from empirical distributions, as in [13]. This topology, while simple, is useful in that such bottlenecks occur frequently in physical networks.

6 Results and Analysis

In this section, I present and analyze experimental results. I begin with comparing the performance of Xavier against the baselines and oracle, as well as against common TCP congestion control algorithms. I then perform an error analysis to guage the relative impact of state-space features and to gain insight into the reward function. All of the following results were obtained by running 10 simulations of each experiment and charting the median throughput and queuing delay exhibited; the ellipses are $1-\sigma$ contours of the maximum-likelihood Gaussian fitting the points. This experimental evaluation methodology was borrowed from [13].

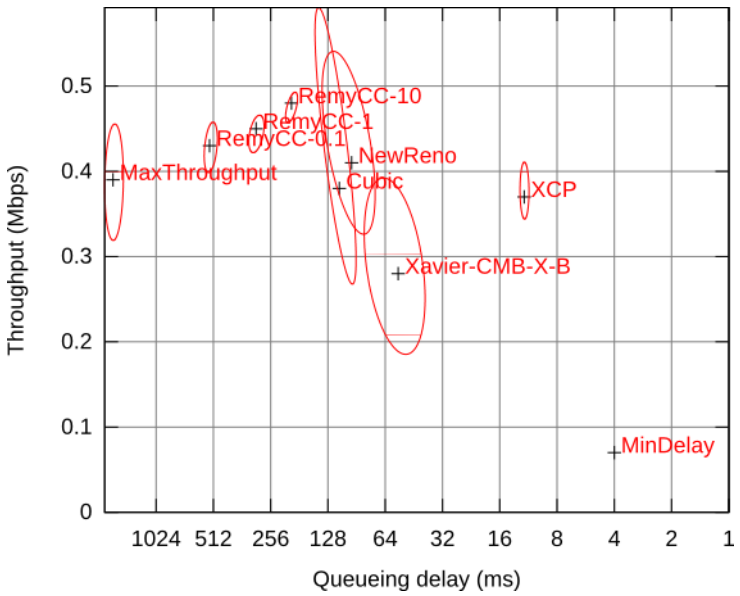

Figure 2: Figure 2. Dumbbell, 3MB Bottleneck.

6.1 Versus Baselines and Oracles

Figure 1 shows the performance of Xavier, the baselines, oracles, and common TCP congestion control algorithms for the dumbbell topology with a 15MB bottleneck switch. The baselines, labeled MaxThroughput and MinDelay, optimize their respective goals at the cost of the maximizing delay and minimizing throughput, respectively, suggesting that the problem of congestion control is relevant here. The ellipse labeled CMB-X-B corresponds to the full Xavier model described in section 4.3, with β tuned to 2.0. The RemyCC models correspond to the oracles, and the numbers following them correspond to the extent to which the model was trained to prioritize throughput – the lower the number, the more throughput was prioritized.

Figure 2 charts the same for a dumbbell topology with a 3MB bottleneck link — notably, the Remy models were not trained for this topology and as such perform poorly, seeing 4-8X more delay than Xavier did. This result is promising because it suggests that, up to a small reduction in throughput, Xavier can generalizes better than Remy does.

6.2 Throughput-Delay Tradeoff in Xavier

The throughput-delay tradeoff in Xavier is controlled through the parameter β. In particular, Figure 3 demonstrates that with $\beta = 0$ (labeled CMB-X), Xavier

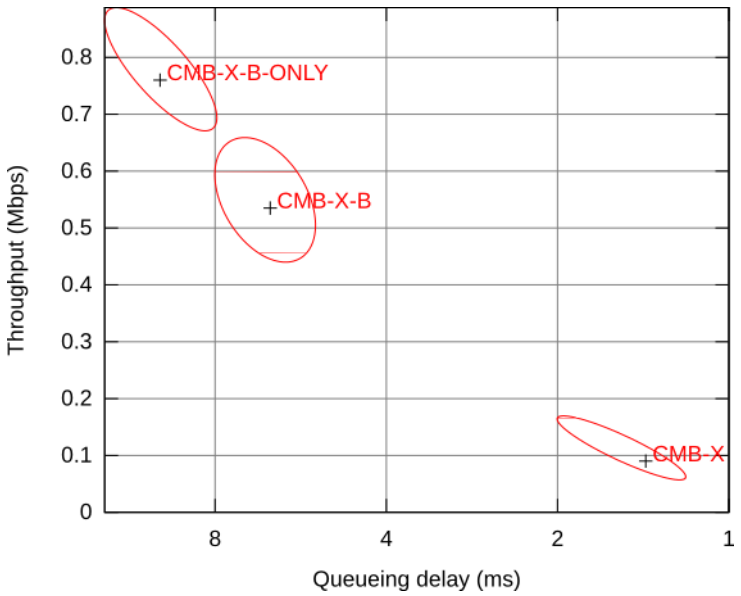

Figure 3: Figure 3. Throughput-Delay Tradeoff in Xavier.

degenerates into the greedy delay policy. Inspecting the weights of the features, the linear decrease action and the states coupled with it had the largest weight — Xavier CMB-X prefered to control the congestion control window, as doing so would minimize delay and thus lead to the largest rewards as controlled by Δ. (Indeed, inspecting feature weights was crucial in tweaking the design of Xavier.) Similarly, fixing $\Delta = 0$ (labeled CMB-X-B-ONLY) led to increased throughput but increased delay as well. That the delay for this variant was still only 12ms was somewhat suprising, but can be explained by noting that smaller delays translate into more frequent ACKs which in turn translate into the accrual of β rewards.

It is unfortunate that the throughput-delay tradeoff has to be controlled by manually tuning the parameter β; non-constant definitions of β were experimented with, such as the percent change in an exponentially weighted moving average of the interarrival time between ACKs, but these metrics were unable to reliably capture the effect of actions upon throughput.

6.3 Relative Importance of Features

Figure 4 charts the performance of Xavier as features were removed from it, one-by-one. Taking variability into account, the models do not differ a great deal; nonetheless, fixing our attention to the medians, there is a trend that adding features tends to increase throughput. In particular, adding EWMA to

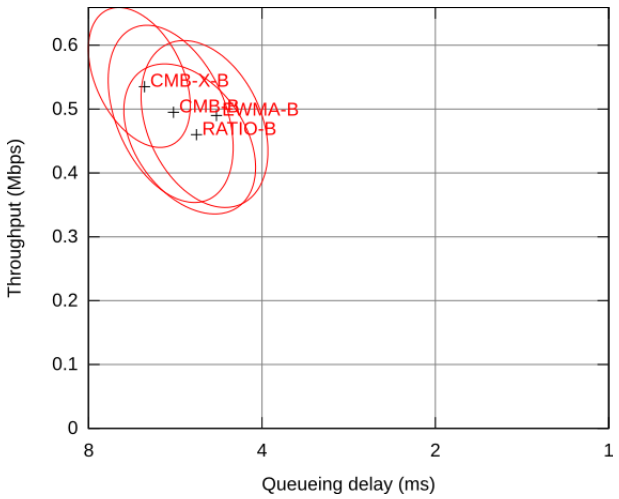

Figure 4: Figure 4. Relative Importance of Features. CMB-X-B is the full Xavier model (CMB short for "combined features"), CMB-B is Xavier without the cross-features, EWMA-B is CMB-B without the RTT-RATIO features, and RATIO-B is CMB-B without the EWMA features

the RATIO model to produce CMB-B and then adding the cross features to produce the full Xavier model both increase the median throughput — both EWMA and the cross features capture historical performance, and the cross features in particular enables Xavier to better relate actions to their effects on congestion and delay, so the increase in throughput makes intuitive sense.

7 Future Work

The experimental results are promising in that they suggest that a reinforcement-learning approach to TCP congestion control is viable — Xavier performs similarly to traditional end-to-end congestional control models and, in the 3MB topology, achieves significantly lower delay. However, as the comparison to Remy in the 15MB experiment shows, Xavier is far from optimal. Future work will focus on (1) designing a reward function more sensitive to the effect of actions on throughput, (2) experimenting further with the state space and using more intelligent feature extractors, and (3) using a non-linear function approximation.

The notion of a reward in a multi-agent system such as this one is noisy because it is difficult to isolate the effects that other agents' actions have upon any given agent's state. That is to say, each agent's state is effected by other agents' actions, making it hard to reliably give out rewards and penalties. As such, it may be necessary to consider models alternative to Q-Learning, or at least

generalizations of Q-Learning to multi-agent cooperative systems such as the one presented in [12].

8 Conclusion

Traditional TCP congestion control, in fixing its mapping from state to control actions, renders itself unable to adapt to changes in network topology and load. This paper proposed Xavier, a first-step towards a more generalizble and flexible model of TCP-based congestion control. Experimental results demonstrate that Xavier performs on roughly par with TCP New Reno and other fixed, end-to-end control algorithms, but underperforms when compared to a hyper-optimized, topology-specific algorithm. Nonetheless, the results are promising: Xavier's performance is acceptable and can only improve with more intelligent reward functions and more sophisticated reinforcement learning algorithms.

References

[1] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pages 671–671, 1994.

[2] S. Choi and D.-Y. Yeung. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. *Advances in Neural Information Processing Systems*, 8:945–951, 1996.

[3] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, pages 317–365, 1998.

[4] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. Pcc: Re-architecting congestion control for consistent high performance. *arXiv preprint arXiv:1409.7092*, 2014.

[5] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40, 2011.

[6] K.-S. Hwang, C.-S. Wu, and H.-K. Su. Reinforcement learning cooperative congestion control for multimedia networks. In *Information Acquisition, 2005 IEEE International Conference on*, pages 6–pp. IEEE.

[7] C. V. N. Index. The zettabyte era–trends and analysis. *Cisco white paper*, 2013.

[8] T. Issariyakul and E. Hossain. *Introduction to network simulator NS2*. Springer Science & Business Media, 2011.

[9] J. Nagle. Congestion control in ip/tcp internetworks. 1984.

[10] L. Peshkin and V. Savova. Reinforcement learning for adaptive routing. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1825–1830. IEEE, 2002.

[11] A. Tarraf, I. W. Habib, T. N. Saadawi, et al. Reinforcement learning-based neural network congestion controller for atm networks. In *Military Communications Conference, 1995. MILCOM'95, Conference Record, IEEE*, volume 2, pages 668–672. IEEE, 1995.

[12] G. Tesauro. Extending q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems*, page None, 2003.

[13] K. Winstein and H. Balakrishnan. Tcp ex machina: Computer-generated congestion control. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 123–134. ACM, 2013.