Experimental Study on different Congestion Control Schemes generated Using REMY

Kishan Varma 111323214

Kedar Sankar Behera 110937490

Kumari Shalini 110451857

Abstract

This paper, we show detailed analysis on different scenarios to answer whether an automated system generated protocol [using Remy] with some prior assumptions, trained and tested on a subset network behavior can be extend to real networks. Will it be able to surpass the performance of conventional manually-created protocols on complex and unpredictable real-time network scenarios? We have also examine and formalize the following concerns through detailed experimental analysis. For example, following concerns as stated by (J. Wroclawski,) 1. Are we performing better by narrowing down the range of operating conditions or is it the ML that chooses best conditions among the desired range of conditions space. 2. Whether Remy suffers from "works today but not tomorrow syndrome". If narrowing down the conditions help then in future when a rare event occurs, will it provide equally good performance?

Code Availability: Our code is available at *https://github.com/matrix-revolution/Experimental-Study-on-different-Congestion-Control-Schemes-generated-Using-REMY*

1 Introduction

The classical congestion-control methods embedded in TCP perform poorly with the new technologies.TCP is not flexible to adapt its congestion control techniques to these new technologies (Winstein Keith,Balakrishnan Hari, 2013). RemyCC is a computer program which can generate protocols based on prior assumptions of the network characteristics. Thses computer-generated protocols has claimed to surpass the performance from those of manually designed congestion-control protocols such as those in TCP-NewReno, TCP-Reno, TCP-Tahoe, TCP-Cubic etc.

Our assumption is that RemyCC may not give an optimal performance on a real-time network with complex network characteristics. The goal of this paper is to compare the performance of various manual congestion control algorithms with that of computer generated on the real time network traffic. Hence, in this paper, we examine couple of different scenarios to test our hypothesis and compare the results. We also tried to reason the cases where the behaviour of Remy differs than expected.

2 Related Work

Since, most of our experiments are influenced from Remy, we have summarized the work given in (Winstein Keith,Balakrishnan Hari, 2013) in this section.

What is Remy? Remy is a computer program that figures out how computers can best cooperate to share a network.It creates end-to-end congestion-control algorithms that plug into the Transmission Control Protocol (TCP). These computer-generated algorithms can achieve higher performance and greater fairness than the most sophisticated human-designed schemes (TCP ex-Machina, 2014).

Why Remy? To understand how Remy works, it's crucial to understand why Remy came in existence in first place. Internet is growing at a rapid rate. New link technologies and sub-networks are emerging and evolved. In the past few years, there has been many new observations where networks characteristics have changed a lot. There have been increase in wireless networks with variable bottleneck rates. Datacenter networks emerged with high rates and short delays. It experienced the condition of bufferbloat

i.e paths with excessive buffering. Even the cellular network experienced variable changes. For example, self-inflicted packet delays, links with non-congestive stochastic loss, and networks with large bandwidth-delay products.

How Remy works? Remy was designed to solve the problem of creating a new congestion-control protocol as the internet advances. It was designed to reduce the human-effort and letting the machine learn and create the schemes to accommodate the network behavior automatically. Congestion-control protocols works on end-points i.e sender-receiver's computer and not inside the network. The performance of such algorithm is governed by the fair allocation of available resources of the network between different users. Hence, Remy defines the objective function that maximizes the score to gain high throughput and low queuing delay. It is trained on a set of assumptions or prior knowledge of the network that a protocol designer need to specify beforehand.

How Remy is designed? The Remy Model contains defining three crucial parts :-

- **Prior Assumptions to the network** -
  - Decide the *design range* of the model
  - Different model with different amount of uncertainity.
  - For example, a protocol designed for Datacenter network will be different than that of protocol designed for cellular network.
  - The very basic model will specify following kind of parameter :-
    * Lower and upper limits of bottleneck link speeds
    * Non-queuing delays
    * Different queue-sizes
    * Various degree of multiplexing.
- **Traffic Model** - Defines how much load is given to the end-points. For example traffic load as seen in following cases :-
  - Web traffic
  - Video conferencing
  - Batch processing
- **Objective Functions** - define objective function that can minimize the loss by maximizing high throughput and low delay score.

How Remy performed? As a result, Remy not only outperformed various manually designed algorithms like TCP Cubic, Compound and Vegas. It also performed well as compared to algorithms that require extensive in-network characteristics like TCP XCP and Cubic-over-sfqCoDel (stochastic fair queueing with CoDel for active queue management). Remy might equally perform well in Datacenter network as well as Cellular Network.

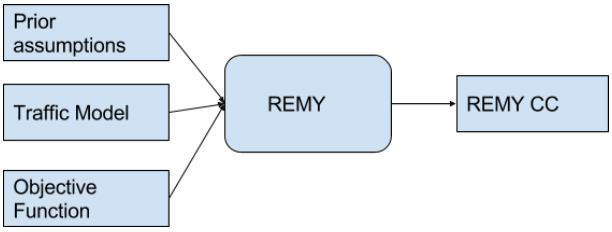


Figure 1: Remy Model

3 Hypothesis

In this section, we are trying to examine following set of hypothesis to understand the importance of various network characteristics and its impact.

- Remy narrows down the network parameter's range.
  - Does the network trained for a specific scenario will outperform the network with more broader range?
  - We replicated this scenario by training our model with wider range of network parameter like increasing the link speed range, number of multiplexing, wider range of queueing delays etc.
- Remy is trained on a specific set of rules which limits to 150. It uses similar concept as Machine Learning classifier, for example, support vector machine for classification task. The only difference is the way objective function is defined which uses throughput and delay to measure the performance at each iteration.
  - Does the Machine Learning learn the correct weights for each network parameters given a specific scenario training examples?
  - Or is it overfitting? That is if model is tested on an unexpected scenario might lead to performance loss.

4 Experiments And Results

Most of our work in this project includes working with NS-2. Here is the brief description about NS-2 followed by the experimentation setup.

4.1 Brief description about NS-2

NS-2 is a event simulator that helps in simulating different types of protocols like TCP, routing, various multicast protocols etc. The simulations could be similar to wired or wireless network. It has been used widely by many for various networks related research. It isn't a fully polished product yet but keep on improving will more usage.

4.2 Setup

Remy works by testing a wide array of possible configurations to arrive at the best possible results. For the experimentation setup, we have followed the following steps :-

- The Remy Code was cloned from `https://github.com/tcpexmachina/remy` repository
- Followed steps from `http://web.mit.edu/remy/` to reproduce the results.

We used the following default configuration to replicate the Remy Figure:

- We are using the dumbbell network with following configurations :-
  - Link-speed = 15Mbps
  - Min RTT - 150 millisecond
- Initially, we ran the whole iteration using 10 parallel threads, it took around 5 days to obtain the output.

For the rest of the experimentation setup, we needed to change the configurations for each scenario. To fasten the process, we made following changes in the code:-

- We needed larger number of simulations to generate the figure.
- By default, there are 10 different simulations runs per data point. We decreased it to 2 simulations per data points
- By default, there are 1000 link speeds spaced between 1 and 1000 Mbps. We divided it into subranges. The following were the ranges:- 1- 50 Mbps, 50 - 150 Mbps, etc
- By default, there are 5 protocols (the 10x, 100x, 1000x, RemyCCs + cubic + cubic/sfqcodel), we also included other protocols like NewReno and Compound as well.
- By default, There are 30 RTTs ranging from 5 to 150 ms. We using only single RTT value. Currently, we tested on RTT = 150 Mbps
- By default, there were 20 duty cycles ranging from 5 to 100. We decreased it to 10.

Hence, initially there were total

$$10 * 1000 * 5 + 10 * (20 + 30) * 5 = 52500$$

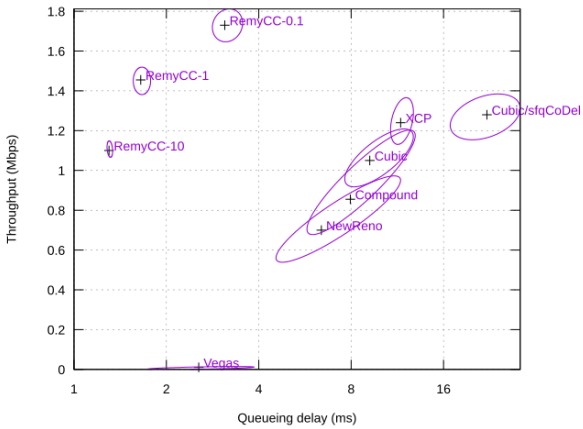
simulations.

We reduced it to

$$2 * 50 * 7 + 10 * (1 + 10) * 7 = 700 + 770 = 1470$$

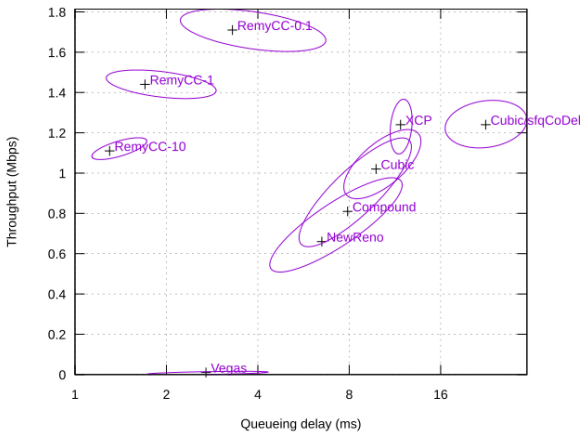
simulations

4.3 Experiment 1: Using Default Configuration



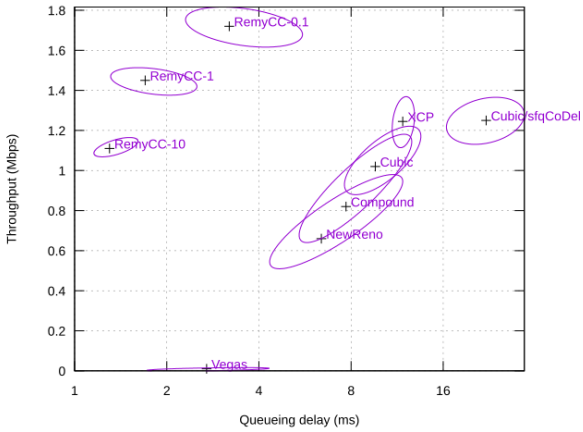
The above figure is generated using Default RemyCC Configurations.

4.4 Experiment 2: Changing number of sources to 16



The above figure is generated by changing the number sources to 16

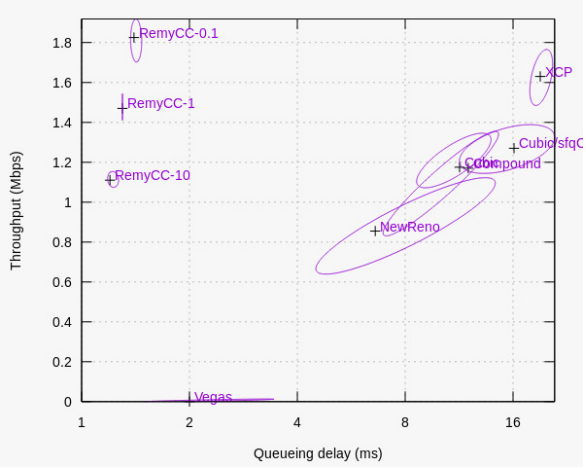
4.5 Experiment 3: Changing number of sources, link-speed, off-process



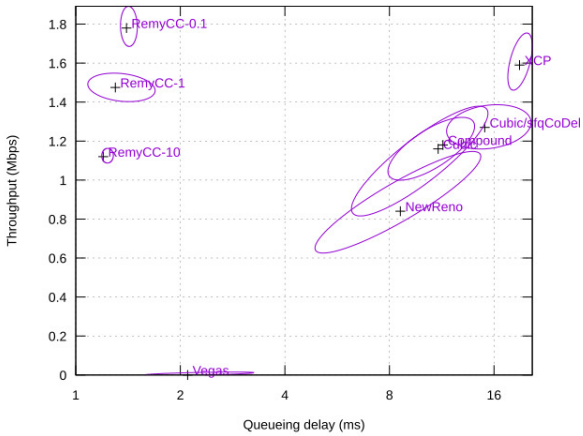
The above figure is generated by changing following network parameters:-

- Num of srcs = 16
- Link Speed from 15 Mbps to 10 Mbps
- Off process from 0.2 to 0.1.

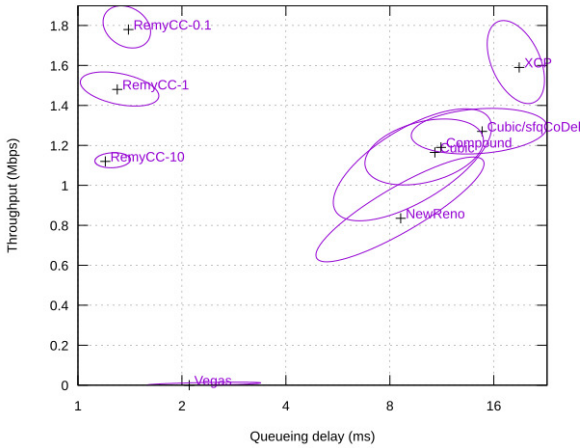
4.6 Experiment 4: Changing number of sources to 4



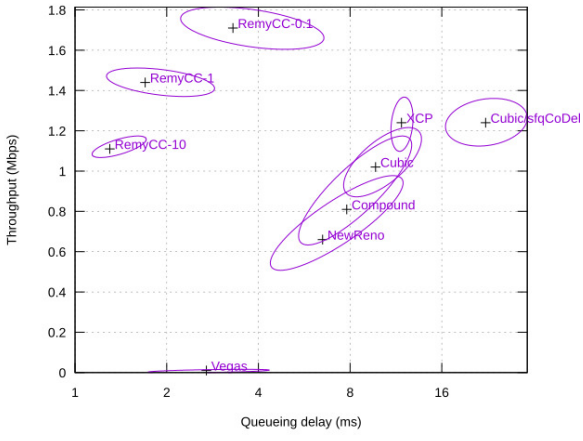
4.7 Experiment 5: Changing number of sources to 4 and link-speed to 100 Mbps



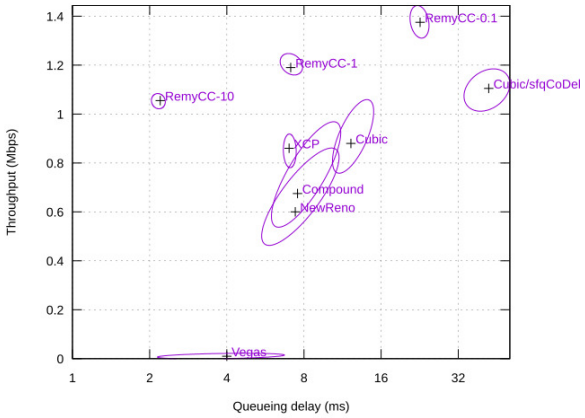
4.8 Experiment 6: Changing number of sources to 8 and link-speed to 32 Mbps



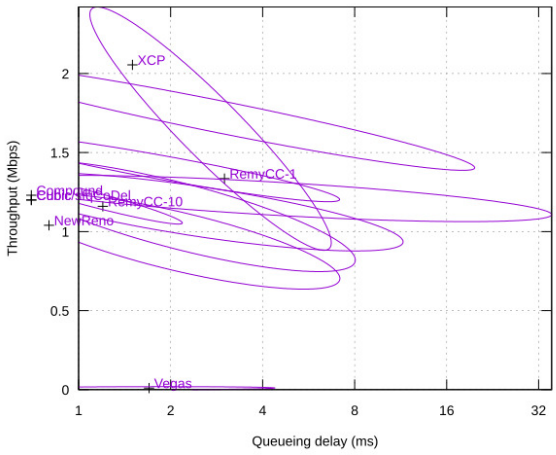
4.9 Experiment 7: Changing number of sources to 2 and link-speed to 500 Mbps



4.10 Experiment 8: Changing number of sources to 2 and link-speed to 1000 Mbps



4.11 Experiment 9: Changing number of sources to 2 and link-speed to 1000 Mbps and max quque size = 10000



5 Evaluation and Analysis

The assumptions provided to Remy at design time, the machine-generated algorithms dramatically outperform existing methods, including TCP Cubic, Compound TCP, and TCP Vegas. As per the experiments conducted by us given in the above graphs ,the general inferences are as follows :-

- Remy outperforms other protocols
- By changing parameters such as the number of sources we get different graphs with greater number of sources having a greater delay variance
- For example, like n = 4 has more convergence than n = 8
- The graph also shows a trend that as the bandwidth product increases the delay variance of the version of Remy with $\delta = 1$
- Here as the bandwidth is varied to 100Mbps and 32Mbps for n = 8 and n = 4 respectively the delay variance increases

6 Discussion

6.1 Observation on different link speed

We ran out model on various link speeds while maintaining the other variables values to default. We used link speeds in range 1 - 1000 Mbps using number of senders = 2. We found that the performance of the model remain consistent upto link speeds 100 Mbps after that there was huge variance in queueing delay from 500 Mbps to 1000 Mbps as seen in Experiment 7 and 8. Although this was the case, Remy still outperformed others with high throughput on whole range of link speed.

6.2 Observation while changing the degree of multiplexing/number of senders

We ran our model on a set of users to determine whether the network performance is impacted when the network topology remains same but the number of users/senders changes. The model was tested using different set of users varying from 1 to 16. We found that as the number of senders increased there was decrease in performance of Remy, though Remy still outperformed other model on varying number of senders. The graphs plotted above using different number of senders while keep the other network parameters constant shows that as the number of senders increase the delay variance in Remy as increases. This shows that the model trained on low degree of multiplexing behaves aggressively when tested for higher degree of multiplexing. This may lead to high queuing delay when we have enough buffer to hold the packets or low throughput when doesn't have enough buffer space and packets are dropped leading to retransmission.

7 Conclusion

In this project, we did a detailed analysis on how the Remy performs given different network conditions. We showed that changing link speed had somewhat modest effect on Remy but as we increased the bandwidth-delay product i.e increas-

ing the queue size as well as the link-speed, the behavior of Remy as well as other protocol changed drastically. This is when, other protocols outperformed Remy. We achieved this performance when link-speed was 1000 Mbps and queue-size was 10,000 packets. We also addressed few other issues which supports why Remy might not give stable performance when trained on small set of network parameters but tested on rare conditions. For this, we trained Remy using imperfect configurations. Then generated various scenarios to evaluate Remy's performance. For example, increased the number of bottlenecks, increased the range of link-speed, increased the degree of multiplexing etc.

8 Acknowledgement

We would like to thank to our Prof Aruna Balasubramanian for her feedbacks throughout the completion of this project. We would also like to thank Prof Keith Winstein, the author of paper Remy without whose help we won't be able to complete our experiment in such a short span of time. We are equally grateful to Anirudh, whose support on understanding nuance of Remy helped us in successful completion of our project.

9 References

[Winstein Keith,Balakrishnan Hari2013] [1] TCP ex Machina:Computer-Generated Congestion ControlKeith Winstein and Hari Balakrishnan Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology, Cambridge http://web.mit.edu/remy/TCPexMachina.pdf

[TCP exMachina2014] [2] TCP ex Machina: Computer-Generated Congestion Control by Keith Winstein and Hari Balakrishnan MIT Computer Science and Artificial Intelligence Laboratory (SIGCOMM 2013) http://web.mit.edu/remy/

[Anirudh Sivaraman, Keith Winstein, 2014] [3] An Experimental Study of the Learnability of Congestion Control : Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan MIT Computer Science and Artificial Intelligence Laboratory (SIGCOMM 2014) http://web.mit.edu/keithw/www/Learnability-SIGCOMM2014.pdf

[J. Wroclawski] [4] TCP ex Machina. J. Wroclawski http://www.postel.org/pipermail/end2end-interest/2013-July/008914.html, 2013.

[5] NS-2 network simulator - http://www.isi.edu/nsnam/ns/