Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

Meta Congestion Control

Semester Thesis

SA-2020-14

Author: Boya Wang

Tutor: Alexander Dietmüller, Maria Apostolaki

Supervisor: Prof. Dr. Laurent Vanbever

March 2020 to June 2020

Abstract

Congestion control algorithm is one of the pearls in network research field. Congestion control algorithm is used to leverage network resources, at the same time, to prevent overloading network. The demand of high-utility network keeps growing, which requires the internet to be low-delay, high-throughput and low-loss. Consequently, a lot of novel algorithms are developed in decades. However, the complexity and diversity of network condition makes it hard to have one single-best congestion control algorithm for all scenarios. Therefore, meta congestion control is emerged as a new idea which allows switching between different algorithms according to network condition in order to obtain better utility. The intuition is that, instead of using a fixed congestion control algorithm, the optimal algorithm is selected for current network condition and usage scenario. This thesis is aimed to explore whether it is beneficial to implement such a meta congestion control and how much the potential gain it can have. It analyzes data collected from a platform consisting of real world nodes to address these two goals.

Contents

Chapter 1

Introduction

1.1 Motivation

Congestion control algorithm plays a crucial role in achieving high network utilization. With growing demand for high bandwidth, higher reliability and lower latency, a lot of novel algorithms have been developed.

However, none of the algorithms is proved to be the silver bullet for congestion control. Congestion control algorithms are usually developed with certain assumptions about network condition and design goals. Some algorithms are designed to deal with long delay, while others works well in low-latency network. In real world, network condition can vary a lot due to the diversity of application traffic, node location, link type... Hence, one algorithm that works well in one scenario may perform badly in another application scenario. The diversity of underlying network and algorithm design procedure lead to the situation where there is no single algorithm which excels in every case.

Intuitively, it can be beneficial to use meta congestion control, which allows to switch between different algorithms in order to have the best application utility for current network condition. Instead of using a fixed default congestion control algorithm all the time, the idea is to select the optimal one for each usage scenario.

There are several challenges to this approach. First, the meta protocol needs to reliably learn the pattern of algorithm performance in order to switch accordingly. The algorithm performance can be affected by many factors that differs for each connection. The impact of various factors are diverse, too. A deeper understanding on how much these factors can affect the performance is needed before implementing meta congestion control. Second, it is not known that how much gain the network utility can get by switching congestion control algorithms. This remains an important question because, if the gain is not considerable, it will be meaningless to implement such a meta protocol.

1.2 Task and goals

The task of this thesis is to investigate the idea of meta congestion control. It takes a closer look at the two challenges mentioned above. First, it addresses whether there is obvious pattern on the choice of "best" congestion control algorithm for certain situation. Second, I evaluate how many benefits can be obtained by using meta congestion control. This thesis aims to serve as the preparation before implementing meta congestion control. It gives a view on whether it is worthwhile changing to such a meta protocol in real world.

1.3 Overview

Chapter 2 provides background knowledge on congestion control and relevant data platform. Chapter 3 describes the whole design of the data analysis part. Chapter 4 shows some highlights of results. Chapter 5 summarizes takeaways and discussions. Additional figures and tables are provided in Appendix A at the end.

Chapter 2

Background

2.1 Congestion control algorithm

Network congestion occurs when a network node needs to process more traffic than it can handle. When a packet arrives at a switch, it is processed based on forwarding rules and forwarded to an output link. A network node or link has a fixed bandwidth. Once packets arrives too fast, network congestion will occur. Typical effects are queuing delay and packet loss. To avoid those negative effects and to leverage network resource, congestion control becomes necessary to network mechanisms.

To achieve congestion control over the network, it is essential to understand the capacity of a connection in order to fully use the resource without overwhelming. As network nodes process thousands of flows every second in a connection, the maximum rate of a connection is limited by the so-called bottleneck link, i.e. the link with the least amount of resource to process the flows along the path. In a path, when data are sent within the bandwidth of bottleneck link, there is no congestion. In this case, the sending rate is equal to the delivery rate. Once it reaches the bottleneck bandwidth, the network fulfills its potential. The sender sends as much data as possible without filling buffers in the intermediate nodes. When the sender rate continues increasing, buffers starts to fill and queue may occur, which will cause extra delay. If the buffer is full, the network node has to drop packets. Congestion control algorithm is to make sure that packets arrive at the receiver at a rate that the bottleneck can support. It detects whether there exists congestion in current network. For congestion-free scenario, it will try to increase the sender rate in order to fully use the bandwidth. If congestion is detected, it will use a more conservative rate to send packets. The key in congestion control is to estimate the current state of network in order to act accordingly.

Congestion control remains an active research field for decades. A large amount of congestion control algorithms are developed and improved by researchers. Algorithms can be classified in to different categories based on which kind of feedback they are used to detect network state.

- Loss-based algorithm

  Loss-based algorithms detect network congestion when buffers are already full and packets start to be dropped. The original congestion control algorithms were mostly loss-based. TCP Reno uses slow start and fast recovery, it is the first loss-based algorithm that was widely deployed[10]. However, its conservative essence of halving the congestion window becomes a problem. Connections were not able to fully utilize the available bandwidth. Hence, other loss-based algorithms were proposed, such as New Reno[9], Highspeed TCP[6] and Westwood[7]. They improved upon Reno by probing for network resource more aggressively, react more conservatively to loss detection and discriminate between different causes of packet loss.

- Delay-based algorithm

  Delay-based algorithms aims to find the point where the queues start to fill. They monitor round-trip time (RTT) and act accordingly. An increase in RTT or a dropped packet indicates network congestion, therefore, the algorithm will reduce sender rate. A stable RTT indicates congestion-free network state. Among these delay-based algorithms, researchers draw great attention on Vegas[2]. Compared to Reno, Vegas includes a modified re-transmission strategy that is based on fine-grained measurements of the RTT as well as new mechanisms for congestion detection during slow-start and congestion avoidance. The design minimizes the queuing delay and helps keep the sending rate stable. Compared to loss-based algorithms, network oscillations are reduced and the overall throughput improves.

2.2 Pantheon platform

Pantheon is a training ground for congestion control research[17]. As an active research field, a lot of new congestion control schemes emerged in the past decades. They were usually tested in academic networks or other small testbeds before deploying across the network. However, it is gradually known that the studies on such academic networks are less likely to generalize the real performance in real diverse network. Another problem is that different congestion control algorithms are tested in diverse test networks. It is hard to compare the results fairly. Therefore, it is difficult to evaluate new ideas and algorithms in a reproductive manner. Pantheon aims to address this problem as a distributed, collaborative system for researching and evaluating end-to-end networked systems.

Pantheon consists of three parts.

- A wrapped software library of congestion control algorithms. It is wrapped and expose the same interface to a full-throttle flow.
- A testbed of network nodes from all over the world.
- A collection of network emulators which are calibrated to match the performance of a real network path.

Pantheon is already used by researchers to develop novel congestion control algorithms like Copa[1] and Vivace[5]. There are other data-driven algorithms that are under developing. However, there is no detailed analysis on its data collection. Especially, it is not clear what is the impact on algorithm performance when node location, data flow direction or link type changes. It leaves more space for exploration on what we could learn from collected data in Pantheon.

Chapter 3

Design

This chapter presents the design of data analysis process. First of all, I clarify the goal of the design and the questions which I would like to answer in this thesis. I then explain the approach to the goal. It could be seen as an overview of analysis procedure. Finally, I elaborate on main design choices. There are several main design ideas mentioned in this chapter: analyzed algorithms, metric, utility function and meta protocol setting.

To be more specific, this thesis intends to explore whether meta congestion control benefits and show quantifiable analysis on how much it may gain from doing so. The whole approach is to first investigate the optimal algorithm under various criteria, second look at potential gain in each case. To implement these two steps, it introduces analyzed algorithms as the ground set of comparison. Metrics and utility score based on application utility are designed as the criteria for algorithm evaluation. Two settings are defined for ”limited switching” scenario and ”omniscient switching” scenario respectively.

3.1 Design goal

The goal of this thesis is to explore whether it is beneficial to use meta congestion control, which enables congestion control algorithms switching according to current network usage condition. Particularly, this thesis focuses on providing quantifiable analysis on potential benefits of meta congestion control. I would like to answer following questions:

- Can applications benefit from switching to different congestion control algorithms based on network condition?
- If so, how much can it gain from meta congestion control? If not, why is it the case?

3.2 Approach

My analysis is based on data from Pantheon[17]. As mentioned before, Pantheon project has collected a large amount of data by running congestion control algorithms in both real world nodes and simulated systems. In this thesis, I only choose the data from real nodes since some researchers have shown that the simulated results could diverge a lot from real scenario results[15]. By doing so, I intend to have a closer look at how congestion control algorithm performs in real world for varied network conditions.

First, it is necessary to derive the definition of the “best” congestion control algorithm. Without this definition, it is hard to compare performance between various algorithms. On the one hand, there are basic metrics which use only one or two measurements of congestion control algorithms. On the other hand, utility scores are defined for the purpose of investigating several application scenarios. This part is to elaborate how to define ”best” under certain usage. Currently, most applications will not specify congestion control algorithm while running. However, it is known that utility functions can be dramatically different from each other, which leaves space for adapting algorithms accordingly. For this part, we would like to know both the scenarios in which the algorithm work well and the scenarios in which it does not show good performance. In other words, we would like to explore not only ”how good it could be” but also ”how bad it could be”.

With the metrics and utility scores in hand, I first investigate if there always exists a fixed best congestion control algorithm globally. There is an underlying assumption of meta congestion control, i.e. there is no single best algorithm for all usage scenarios. Since, if there exists such a globally optimal choice, it could be more reasonable to stick into this single algorithm. If there is no such algorithm, there will be more interesting questions that we may ask further. Therefore, I use data from Pantheon to examine this assumptions in order to build the basis of further exploration.

Furthermore, I examine whether there exists a fixed best algorithm when aggregating within different clusters, e.g. measurement node location, data flow direction, link type... By doing so, we can gain more insights on when and how frequently it should switch to other algorithms. At the same time, we would know for which percentage of measurements one algorithms performs the best. In order to address the different needs for various usages, I use both basic metrics derived from delay, throughput, loss measurements, and utility scores.

3.3 Overview of analyzed algorithms

Pantheon runs a collection of algorithms on test nodes around the world. The analyzed algorithms have divergent design goals. Here is a short introduction on each congestion control algorithm.

TCP Cubic was first derived as a TCP variant for high-speed network environments[8]. It calculates window growth by a cubic function with elapsed time since the last loss event. This cubic function provides a good stability and scalability, and thus, improves TCP-friendliness and RTT-fairness.

QUIC Cubic is the implementation of Cubic in QUIC protocol. QUIC is an internet transport protocol which intends to be a replacement for TCP[11]. It is used by more than half of all connections from the Chrome web browser to Google’s service[16]. It aims to have much-reduced latency than a TCP connection.

Copa is a delay-based congestion control algorithm[1]. Its design aims to have lower queuing delay with additional mechanisms to deal with drawbacks when it shares the network with other loss-based algorithms.

TCP Vegas also emphasizes packet delay[2]. This algorithm depends on accurate estimation on RTT value. While it shows less loss compared to other algorithms (e.g. Reno), its performance wanes as it shares the internet with other congestion control algorithms.

Verus is designed for cellular channel[18]. The main idea is to continuously learn a delay profile that reveals the connection between packet delay and window size over short epochs. It uses this kind of connection to adjust the window size. In addition, it keeps standard TCP features including decrease upon packet loss and slow start.

The author of [3] argues that the design choice of loss-based TCP congestion control results problems in the network. Not all packet loss should be interpreted as “congestion”. Thus, BBR is developed to address this problem.

PCC-Allegro[4] and PCC-Vivace[5] are two algorithms under the same PCC framework. PCC stands for performance-oriented congestion control. Most TCP variants uses a hardwired mapping rate control architecture which links certain predefined packet-level events to certain predefined control responses. PCC runs an online learning algorithm to dynamically choose the sending rate as the data stream.

Indigo[17] and FillP are two ongoing congestion control algorithms that use Pantheon data in design. Pantheon's website shows that Indigo is a data-driven algorithm while there is no more information for FillP. It remains unclear that how they are designed but they are widely tested on Pantheon.

Algorithms	Congestion Signal	Additional Feature	Label
TCP Cubic	loss		TCP Cubic
QUIC Cubic	loss		QUIC Cubic
Copa	delay		Copa
TCP Vegas	delay		Vegas
Verus	delay, loss	cellular-oriented	Verus
TCP BBR	delay, loss		BBR
PCC-Allegro	delay, loss		PCC
PCC-Vivace	delay, loss	data-driven	Vivace
Indigo	unknown	data-driven	Indigo
FillP	unknown		FillP

Table 3.1: List of analyzed algorithms with relevant information. Some are loss-based, others are delay-based. There are also algorithms designed by using both delay and loss. For the purpose of simplicity, the algorithm is referred by its label in the thesis.

3.4 Basic Metric

Pantheon provides three measurements for algorithm tests: 95% delay, throughput and loss rate. I use these three measurements to interpret algorithm performance in a metric. Notice that there is no single metric that can contain all concerns of applications. Generally, applications require congestion control algorithm to have low delay, high throughput and low loss rate. However, it is usually not realistic to have the three characters at the same time since it exists trade-off among the three. In addition, requirements may vary in different scenarios. In this thesis, I provide four kinds of basic metrics (Table 3.2), i.e. delay-only metric, throughput-only metric, loss-only metric and a combination of delay and throughput. In the last metric, I combine delay and throughput as it does in a version of Kleinrock's power metric. The main idea is to obtain both low delay and high throughput. It does not take loss rate into consideration since packet loss does not reflect congestion control algorithm performance in some situations[15].

3.5 Utility function

Although basic metrics using delay, throughput and loss rate are widely used in measuring performance of congestion control algorithms, there are circumstances where there exists other restrictions in specific application. For example, in file transfer, it is reasonable that it only requires a high throughput, and it does not care much about delay. In this scenario, the throughput-only metric satisfies its need. However, in VoIP, there is an upper bound of delay with other restrictions on

Metric	Definition of metric
Delay-only metric	-log(95% delay)
Throughput-only metric	log(throughput)
Loss-only metric	-log(loss rate)
Delay-and-throughput metric	log(throughput / 95% delay)

Table 3.2: Metric design using basic measurements.

throughput and loss rate. Once the delay is too large, the quality of VoIP call deteriorates because users can hardly hear or response each other in a timely manner. None of the four basic metrics meets the need in such scenario. In other words, the four basic metrics may not be sufficient to examine algorithm performance in applications whose utilities have non-linear changes[12, 13, 14]. Therefore, I develop utility functions for such applications to have a better understanding on how well congestion control algorithms work.

In this thesis, I choose three typical applications and develop corresponding utility functions in order to rate the performance of algorithms. The author of [13] shows practical delay and loss rate boundaries of VoIP, video streaming and online gaming. As is shown to us, the upper bound of delay and loss indicates that, once the real measurement is above the threshold, the utility of this application will become unacceptable to the user. On the contrary, once the real measurement is below the lower threshold, this measurement becomes less important while examining the application utility. Besides, it is noticed that different applications have divergent tolerance on delay and loss.

Application	Delay (ms)		Loss (%)	
	min	max	min	max
VoIP	100	150	1	3
Video Streaming	200	1000	0.01	1
Online Gaming	50	150	1.5	3.5

Table 3.3: Empirical thresholds of VoIP, video streaming and online gaming. To satisfy the utility of application, the real measurement must below the upper bound, otherwise, there will be dramatic decline on utility. The lower bound indicates that once the measurement is below this boundary, this measurement will not affect the utility of this application considerably[13].

Given those application-specific requirements, I design the utility function of applications in a way such that it can reflect the thresholds. I define *Utility Score* using three parts, i.e. *Delay Score*, *Throughput Score*, *Loss Score*. Each part is in range $[0, 1]$. After normalization, the range of *Utility Score* is also $[0, 1]$

$$UtilityScore = \frac{DelayScore + ThroughputScore + LossScore}{3} \tag{3.1}$$

As is shown in Figure 3.1, the higher the utility score is, the better the utility of the application is. As for *Delay Score* and *Loss Score*, if the real measurement is beyond the upper bound, *Utility Score* will be set to 0. Because the network condition cannot support the application, the user experience is unacceptable. On the contrary, if the score reaches the lower bound, the application works well. But the user experience will not change considerably even when the measurement continues improving. There is no obvious threshold for throughput in the three applications.

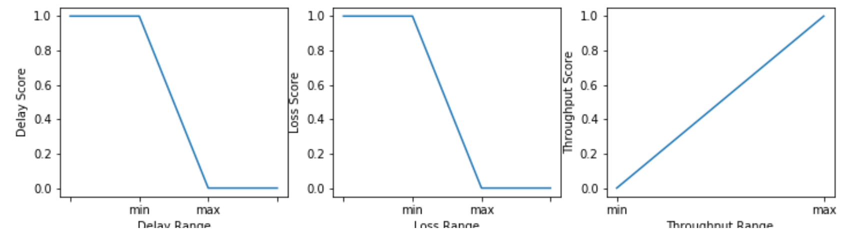


Figure 3.1: Delay Score, Throughput Score and Loss Score.

Hence, I use the minimum and maximum of real throughput measurements to normalize the score. Intuitively, application prefers higher throughput which could support higher resolution or better voice quality.

3.6 Meta protocol setting

In order to quantify the potential benefit of meta congestion control, I define two meta protocol settings for comparison. One is static meta protocol, the other is dynamic meta protocol. In static meta protocol, a *static best algorithm* is selected. And the connection keeps using this algorithm during corresponding measurements. In dynamic meta protocol, it gives an omniscient view of all algorithm performance and it can dynamically switch to any algorithm at any time. A *dynamic best algorithm* is defined here to represent the best-possible algorithm it can have.

Static best algorithm is defined as the single congestion control algorithm which is ranked in the first place for the most times through all test measurements. As its name shows, this best algorithm is "static", which means there can be only one algorithm chosen as the static best algorithm and no switching is possible during all the measurements. The idea is to choose the "best" congestion control algorithm during all measurements and use it as a static-optimal option. It is reasonable and practical for real scenario. For example, user intend to use VoIP to have a one-hour-long call. A naive way to choose the optimal congestion control algorithm is to use the one which performs the best for the most times in all measurements in this scenario. Once it finds the static best algorithm, it uses this algorithm for the whole call. Even though it is possible that, sometimes, there exist other algorithm performs better than this static algorithm in some measurements, it still keeps using the algorithm and refuse switching.

Dynamic best algorithm is defined as the congestion control algorithm which is ranked in the first place in this test measurement. As its name shows, dynamic best algorithm may not be a single algorithm for all test measurements. Instead, it is "adaptive" for each measurement. This setting is aimed to find the most potential gain in the case where user is allowed to switch to any algorithm at any time. In other words, it is assumed that the user has omniscient view and always choose the best algorithm under a certain criterion in all measurements. Apparently, dynamic best algorithm is the best possible choice it may have in congestion control. There is no way to go beyond this omniscient best algorithm.

Given static meta protocol setting and dynamic meta protocol setting, it is able to compare both of them with TCP Cubic to obtain the gain in basic metrics or utility score. TCP Cubic is widely implemented and used by default in both Linux kernels and Windows. Hence, it provides a good basis for comparison. The advantage over TCP Cubic is defined as the substitute of the score of chosen algorithm with the score of TCP Cubic. If the advantage is positive, it means the chosen algorithm performs better under this criterion. Otherwise, the chosen algorithm performs worse than the default algorithm TCP Cubic.

Chapter 4

Data Analysis

In this chapter, data analysis results are presented. There are two main sections. The first section shows that there is no single-best algorithm in congestion control. I choose a random test node in Pantheon and examine algorithm performance under basic metrics. It then becomes the basis of the discussion because it indicates that switching between different congestion control algorithms can be beneficial. Therefore, the second part of this chapter focuses on aggregation for further evaluation. It gives us more insights on what algorithm performs well in which percentage of the measurements. For plots and tables which are not presented in this chapter, please refer to Appendix A at the end of the report.

4.1 No globally optimal algorithm

There exists an underlying assumption in meta congestion control: there is no globally optimal congestion control algorithms. Since if there is a single-best algorithm, it makes no sense to consider switching to other algorithms. Because it will never improve the network performance in this case. Therefore, this assumption is the basis of the idea of meta congestion control. I choose a random test node in pantheon to validate this assumption.

As hypothesized, no single algorithm outperforms all others all the time in all metrics. For example, on the test node in Mexico, Indigo and Vivace take turns for best performance while evaluating delay-only metric (Figure 4.1). It also shows that the delay of algorithm varies through the time. In addition, algorithm ranking varies for different metrics. For instance, Vivace is ranked as the best for low delay (Figure 4.1) but it has low throughput as well (Figure A.1). It highlights the potential benefit of meta congestion control. In other words, if the need changes (e.g. from low delay to high throughput), switching congestion control accordingly is reasonable. Third, there exists cases where one algorithm prevails the other algorithms in all measurements under a metric (Figure A.3). It shows us that switching to other algorithms may not be helpful in some cases. Hence, it is important to know in which scenario the meta protocol should switch while implementing meta congestion control in real network.

4.2 Aggregation

In this section, I evaluate congestion control algorithms for each variable to gain better understanding on algorithm performance. By doing aggregation, the data will not be treated as time series. Instead, I focus more on statistical characters of data and use *Cumulative Distribution Function*

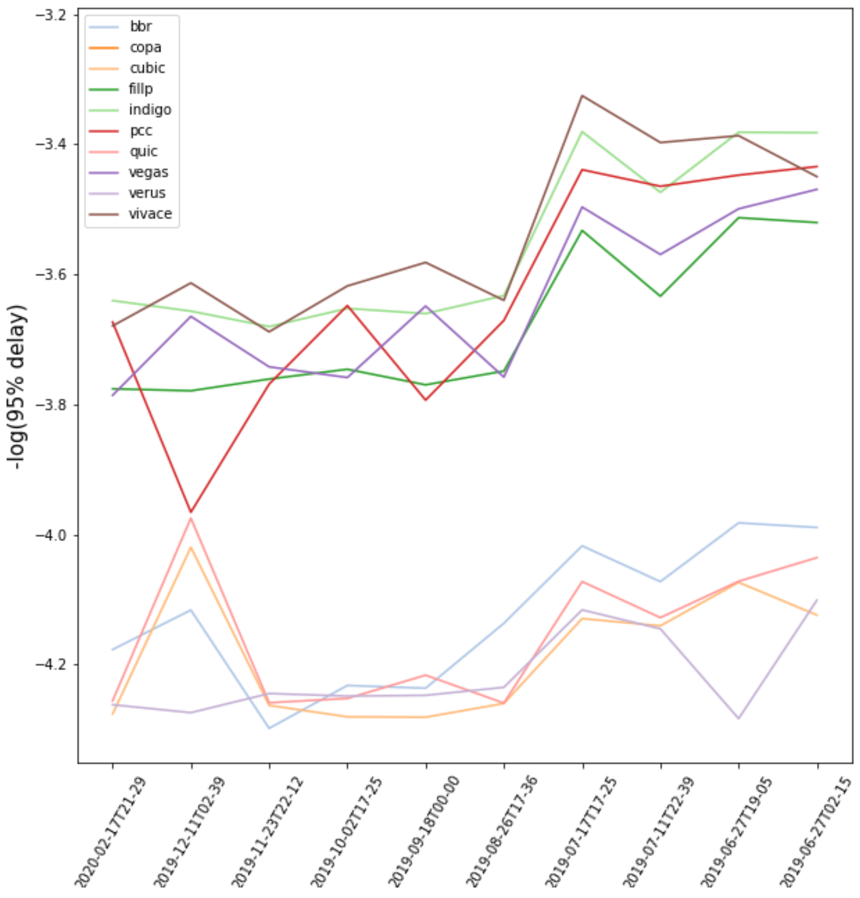


Figure 4.1: Congestion control algorithms performance using delay-only metric. Data are from a test node in Mexico. The x axis is the time of measurement in the form of “year-month-date T hour-minute”. The y axis is generated using -log(95%delay).

(CDF). The purpose is to understand that, given the information that there is no globally optimal algorithm, which algorithm performs the best in which percentage of the tests.

In each subsection, I first use four basic metrics and three utility scores to evaluate algorithms. Then I plot CDF in three applications to further examine congestion control algorithm performance separately. The goal is to see not only ”how well the algorithm performs” but also ”how much loss the algorithm has in worse cases”.

4.2.1 Source node

In this part, the source node is fixed to evaluate congestion control algorithms. Each source node is evaluated respectively, i.e. London, Iowa, Tokyo, Sydney.

For delay-only metric, QUIC Cubic shows dominant performance. It has the lowest delay for at least 85% measurements for all four source nodes (Table A.1, A.2, A.3, A.4). For three among the four source nodes, it is ranked as the best algorithm for more than 90% measurements. In this case, keeping using QUIC Cubic can be a good choice. Additionally, the source node does not show considerable effect on algorithm choice.

For throughput-only metric, FillP is the congestion control algorithm that has the highest throughput for most measurements. For Sydney source node, it is the best one for 77.08% measurements (Table A.4). It is noticed that FillP is the all-measurement-best algorithm in both Iowa (Table A.2) and Tokyo (Table A.3). If throughput is the only concern, switching to other algorithm is less promising here. The location of source node is not an important factor, either.

For loss-only metric, source node location affects the performance of algorithms. Indigo has the lowest loss for 33.33% measurements in London (Table A.1). It means, for the other 2/3 measurements, Indigo does not have the lowest loss. Using meta congestion control can be beneficial for 2/3 measurements. Combining both delay and throughput, FillP has the best performance for at least 85% measurements in four source nodes. It leaves less space for benefiting from switching congestion control algorithm.

Figure 4.2 is presented here as an example on how to interpret the plot. For VoIP, the static best algorithm is Vegas. In 30% measurements, Vegas performs worse than TCP Cubic. For half measurements, Vegas performs better than TCP Cubic with advantage more than 0.2. For dynamic best algorithm, its advantage is above 0.4 for more than 40% measurements. For video streaming, the static best algorithm is also Vegas. In 40% measurements, Vegas performs worse than TCP Cubic. It shows advantage above 0.2 for only about 20% measurements. For dynamic best algorithm, its advantage is less than 0.2 for about 80% measurements. However, it shows more than 0.4 gain in 20% measurements. For online gaming, Vivace becomes the static best algorithm. For more than 40% measurements, it performs worse than TCP Cubic. Its advantage is above 0.2 for more than 40% measurements. For dynamic best algorithm, it shows more than 0.2 advantage for 60% measurements.

As is shown in Figure 4.2, the CDF plot of static best algorithm and the CDF plot of dynamic best algorithm have similar trend. Although there are cases where the static best algorithm has worse performance, the loss is no larger than -0.1 for all measurements. It is noticed that the static best algorithms are different for three applications. Hence, switching according to application scenario is reasonable. However, switching dynamically for each measurement does not gain much more benefit over static best algorithm.

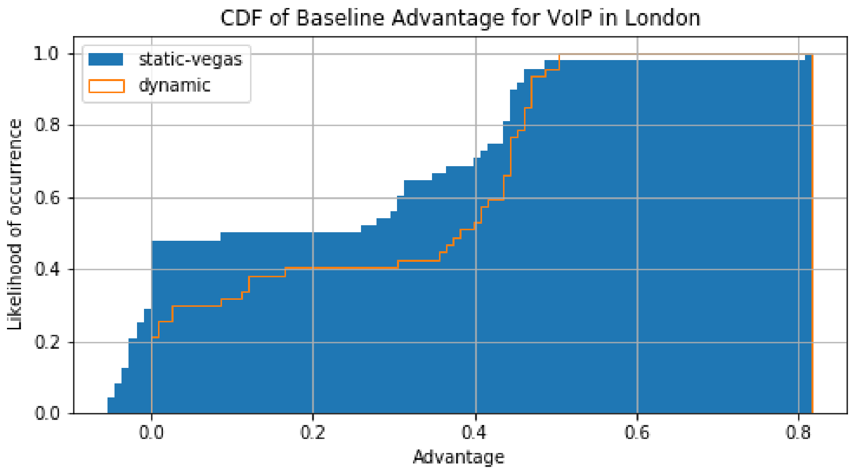
However, the CDF plot in Iowa does not have similar trend as it does in London (Figure 4.3). For video streaming, the static best algorithm is FillP. An interesting thing is that FillP shows loss larger than 0.6 for about 40% measurements when compared to TCP Cubic. In some circumstances, FillP even shows loss above 0.8. Notice that all utility scores are between $[0, 1]$. It means FillP has dramatic performance drop in some cases. Therefore, dynamic switching is necessary in order to avoid the large loss of static best algorithm in Iowa source node.

Figure 4.4 presents results in Sydney for video streaming.TCP Cubic is the static best algorithm. As for the dynamic best algorithm, it shows advantage above 0.2 for only 10% measurements. Since TCP Cubic as the baseline algorithm has the best utility for the most measurements, users can keep using this baseline for video streaming traffic.

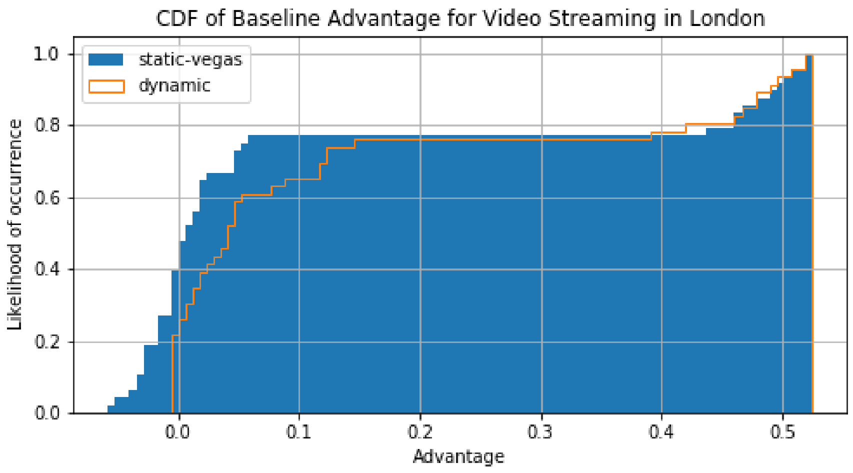
4.2.2 Destination node

In this subsection, we evaluate congestion control algorithms in two destination server nodes, i.e. California and Brazil. It is shown that destination node location is an important factor on the choice of static best algorithm. FillP and Indigo outperform the other algorithms in California destination node, while TCP Cubic and Vivace are the winners for Brazil destination node.

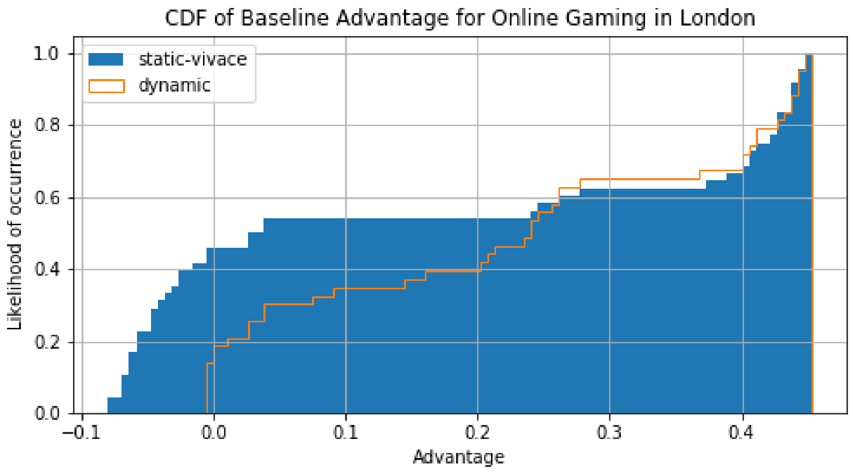
Table 4.1 shows the static best algorithm with its percentage in California destination node.



(a)



(b)



(c)

Figure 4.2: CDF for application utility in London

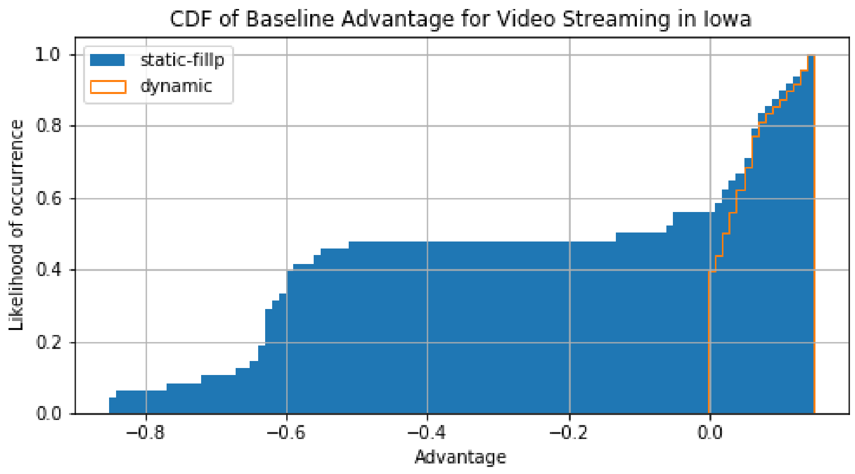


Figure 4.3: CDF for application utility in Iowa for video streaming

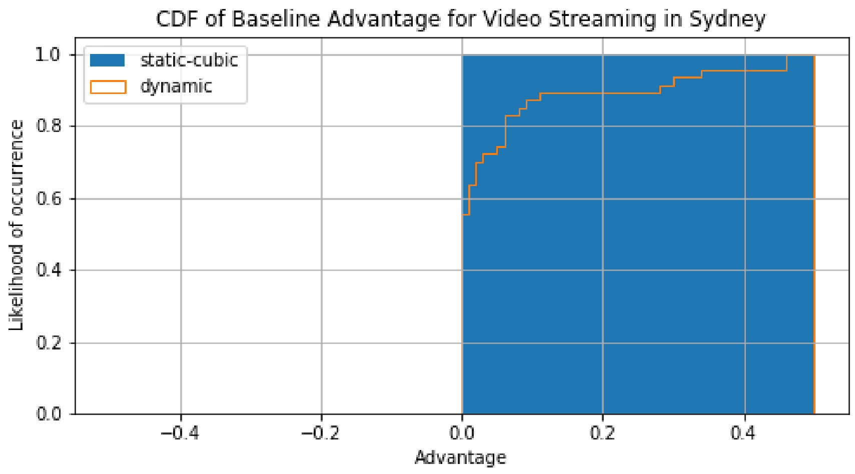


Figure 4.4: CDF for application utility in Sydney for video streaming

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	29.69
Throughput-only	FillP	65.63
Loss-only	Indigo	29.69
Delay-and-throughput	Indigo	56.25
VoIP	FillP	42.19
Video Streaming	Indigo	31.25
Online Gaming	FillP	46.89

Table 4.1: Static best algorithms for different metrics in California destination node.

FillP and Indigo are chosen the most times in all the algorithms. An observation is that, although QUIC Cubic is selected as the static best algorithm again for delay-only metric, it shows the best performance for only 29.69% measurements. In other words, it is not the best option for more than 2/3 measurements. Actually no algorithm performs the best for a portion larger than 1/3 if we only evaluate the delay.

Metric	Static Best	Percentage (%)
Delay-only	Vivace	34.38
Throughput-only	TCP Cubic	46.88
Loss-only	Vivace	53.13
Delay-and-throughput	Vivace	43.75
VoIP	TCP Cubic	50.00
Video Streaming	Vivace	43.75
Online Gaming	TCP Cubic	50.00

Table 4.2: Static best algorithms for different metrics in Brazil destination node.

Table 4.2 shows static best algorithm algorithms in Brazil destination node. It is noticed that TCP Cubic and Vivace are the winners. TCP Cubic performs the best in about half of all measurements for throughput-only metric, VoIP and online gaming. Vivace beats the other algorithms for delay-only metric, loss-only metric, delay-and-throughput metric and video streaming for 34.38%, 53.14%, 43.75% and 43.75% measurements respectively. Compared to the results in California destination node, the choice of static best algorithm is different in this Brazil destination node.

Figure A.7 shows the results in three application scenarios for California. For VoIP application, the static best algorithm is FillP. One the one hand, FillP shows great loss (more than 0.8) for 20% measurements compared to TCP Cubic. On the other hand, FillP shows comparative utility performance with TCP Cubic for the rest measurements. Remind that all utility scores are in range $[0, 1]$, we should pay attention to this kind of dramatic performance drop if we choose congestion control algorithm in "static" setting. It means, although FillP shows slight advantage in some cases, the utility of VoIP using FillP could be unacceptable in a considerable time (in this case, 20%). The dynamic best algorithm does not show advantage above 0.2 in all measurements. For video streaming, Indigo becomes the static best algorithm. However, it performs worse than TCP Cubic for half of all measurements, with only 10% measurements in which it shows advantage above 0.2 over TCP Cubic. The dynamic best algorithm is not promising as well, since it shows advantage above 0.2 for only 10% measurements. For online gaming, FillP becomes the static best algorithm again. It shows similar character as it does in VoIP. That being said, FillP has dramatic loss more than 0.8 compared to TCP Cubic for 20% measurements.

Figure A.8 shows congestion control algorithm performance in Brazil test node. For VoIP, the static best algorithm is TCP Cubic. The dynamic best algorithm shows advantage above 0.2 for 40% measurements. For video streaming, Vivace is the static best algorithm. For 50% measurements, it performs worse than TCP Cubic. In addition, its advantage is in range $[0, 0.2]$ for almost all measurements. The dynamic best algorithm shows similar performance. Its advantage also concentrates within the range $[0, 0.2]$. For online gaming, TCP Cubic is again the static best algorithm. The dynamic best algorithm is not really promising in this scenario, because its gain over TCP Cubic is within range $[0, 0.2]$ for 90% measurements.

4.2.3 Node pair

I examine congestion control algorithms by fixing node pair in this subsection. Notice that there are two kinds of node pairs in Pantheon data. The first one is node-to-closest-cloud, the second one is cloud-to-cloud. Here we choose node pairs from both categories and analyze them respectively. Compared to the analysis we have done before, fixing node pair means both source node and destination node are determined. It gives more restriction on node location compared with the case where just source node or destination node is fixed. There are seven node pairs that are investigated here, the first three of them are node-to-closest-cloud tests, the other four pairs are cloud-to-cloud tests.

Stanford-AWS California

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	37.50
Throughput-only	FillP	75.00
Loss-only	Vegas	29.17
Delay-and-throughput	Indigo	20.83
VoIP	FillP	33.33
Video Streaming	BBR	31.25
Online Gaming	FillP	33.33

Table 4.3: Static best algorithms for different metrics in Stanford-AWS California node pair.

Table 4.3 presents evaluation results for metrics from Stanford to AWS California. It is noticed that FillP has the highest throughput for 3/4 measurements. However, for the other metrics, none of the static best algorithm prevails the other algorithms in more than half of the measurements. It means that the application cannot benefit from switching to other congestion control algorithm for more than half of the time. It remains to show whether dynamic setting can increase application utility.

However, the dynamic best algorithm has large loss in the worst measurements while it has only slight advantage over TCP Cubic for some measurements (Figure A.9). For VoIP, FillP is the static best algorithm. However, it performs worse than TCP Cubic with loss more than 0.8 in 30% measurements. At the same time, the dynamic best algorithm does not show advantage above 0.2 for all measurements. For video streaming, BBR becomes the static best algorithm. In only half of the measurements, BBR shows comparative or better performance than TCP Cubic. Note that BBR shows small advantage within $[0, 0.1]$ for all measurements. As for the dynamic best algorithm, its performance is also only slightly better (less than 0.1) than TCP Cubic for all measurements. For online gaming, FillP is chosen as the static best algorithm. It shows similar performance as we saw in VoIP. While there are dramatic drop on the performance for 30% measurements, the advantage is not really considerable.

Mexico-AWS California

Compared to various static best algorithms in Stanford-AWS California node pair, the situation in Mexico-AWS California node pair is more consistent. Table 4.4 shows static baseline for different metrics from Metrics and AWS California. For delay-only metric, Vegas is selected as the static baseline. For all the other metrics, Indigo is selected. It shows extremely good performance for delay-and-throughput metric and video streaming (90.63% and 93.75%). It seems that using Indigo instead of TCP Cubic is a reasonable choice for this node pair. Before making this conclusion, it is necessary to further investigate whether the benefit is large enough for most measurements.

Metric	Static baseline	Percentage (%)
Delay-only	Vegas	37.50
Throughput-only	Indigo	46.88
Loss-only	Indigo	43.75
Delay-and-throughput	Indigo	90.63
VoIP	Indigo	56.25
Video Streaming	Indigo	93.75

Table 4.4: Static baseline algorithms for different metrics in Mexico-AWS California node pair.

Figure A.10 shows results for node pair from Mexico to AWS California. For VoIP, Indigo becomes the static baseline. For 20% measurements, it performs worse than TCP Cubic, but it is noticed that the loss is extremely small (less than 0.025). For the other 80% measurements, it obtain a gain less than 0.1. The dynamic baseline does not show large gain, either. Its advantage is within 0.1 for all measurements. In this case, we could say that TCP Cubic performs relatively well as a default option. Even with omniscient view in meta congestion control, the potential gain is limited. For video streaming, the situation is Indigo seems to be both the static baseline and dynamic baseline for all measurements. For half of all measurements, the gain is within 0.1, while, for the other measurements, the advantage is between 0.1 to 0.3. For online gaming, Indigo is also selected as static baseline. For all measurements, it shows advantage over TCP Cubic. For half of the measurements, it has advantage within 0.1, for the other half, its gain is between 0.3 and 0.4.

For all three applications, the static best algorithm and the dynamic best algorithm shows similar distribution of CDF. In omniscient setting, the dynamic best algorithm increases application utility within 0.1 in at least half of the measurements. In the other half measurements, the advantage of dynamic best algorithm varies for each application. For VoIP, it is fine to use default algorithm (TCP Cubic) since the advantage is negligible. For online gaming, the advantage of meta congestion control can be larger than 0.3 in half of the measurements.

Brazil-AWS Brazil

Metric	Static Best	Percentage (%)
Throughput-only	TCP Cubic	90.63
VoIP	TCP Cubic	90.63
Online Gaming	TCP Cubic	90.63

Table 4.5: Static best algorithms for different metrics in Brazil-AWS Brazil node pair.

TCP Cubic shows great performance in Brazil-AWS Brazil node pair (Table 4.5). It is the best algorithm for throughput-only metric, VoIP and online gaming in more than 90% measurements. In these three cases, switching from TCP Cubic becomes less promising.

Furthermore, the CDF plot shows that the potential gain of using meta congestion control is negligible, too. Figure A.11 shows the result in Brazil node pair. For VoIP, Cubic performs extremely well. It is selected as the static best algorithm. In addition, the dynamic best algorithm can hardly prevail TCP Cubic for all measurements. For video streaming, Vegas is the static best algorithm. It shows that the static best algorithm has similar performance as TCP Cubic does in this scenario. As we can see, the difference between them is within range $[-0.06, 0.03]$. The dynamic best algorithm does not have considerable advantage for all measurements, either. For online gaming, the situation is similar to the one we have in VoIP. TCP Cubic is the static best algorithm, and even the dynamic best algorithm shows almost the same performance as TCP Cubic.

London-Iowa

In London-Iowa node pair, FillP is selected as the static best algorithm for several metrics in more than half of the measurements. It even has the highest throughput for all measurements. Considering that the time range of all measurements is more than one year, the result is astonishing. In this scenario, switching to FillP is beneficial for applications.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	93.75
Throughput-only	FillP	100.0
Loss-only	QUIC Cubic	56.25
Delay-and-throughput	FillP	75.00
VoIP	FillP	81.25
Video Streaming	FillP	68.75
Online Gaming	Indigo	50.00

Table 4.6: Static best algorithms for different metrics in London-Iowa node pair.

CDF plot also shows that meta congestion control is beneficial for this node pair. Figure A.12 shows the result for London-Iowa node pair. For VoIP, FillP is the static best algorithm. Compared to TCP Cubic, FillP's performance varies a lot. As is shown in the plot, the difference is in range $[-0.6, 1.0]$. For 80% measurements, FillP beats TCP Cubic. For the other 20% measurements, it performs worse than TCP Cubic. For more than half of the measurements, FillP has advantage above 0.2. As for the dynamic best algorithm, its advantage concentrates in two ranges, i.e. $[0, 0.2]$ and $[0.4, 0.6]$. For video streaming, FillP is the static best algorithm. One interesting thing is that, although FillP has advantage for 80% measurements, it also shows loss for more than 0.8 in about 10% measurements. As for online gaming, the static best algorithm is Indigo. For half measurements, Indigo shows advantage above 0.2 over TCP Cubic. The dynamic best algorithm shows gain over all measurements.

Iowa-Tokyo

Table A.6 shows similar pattern as we have seen in London-Iowa node pair. FillP still shows great performance for several metrics. For detailed results, please refer to the appendix. It seems that switching to FillP can be beneficial. To validate this conclusion, CDF plots are needed to help understand how much gain application utility can have.

It shows that the static best algorithm has huge loss in the worse cases while the dynamic best algorithm does not have large gain over TCP Cubic in the best cases. Figure 4.5 is used as an example. For video streaming, the static best algorithm is FillP. It has loss larger than -0.8 for 20% measurements. For 50% measurements, FillP shows advantage over TCP Cubic. The dynamic best algorithm has advantage within range $[0, 0.2]$ for all measurements. Therefore, meta congestion control can be helpless in this node pair.

Tokyo-Sydney

FillP shows dominant performance for several metrics in this node pair, too. Table 4.7 shows part of the results. Notice that FillP has the highest utility for online gaming in 93.75% measurements. After taking a closer look at the CDF plot (Figure 4.6), FillP does not have advantage above 0.2 for all measurements.

Figure 4.6 shows the other results in Tokyo-Sydney node pair. For VoIP, the static best algo-

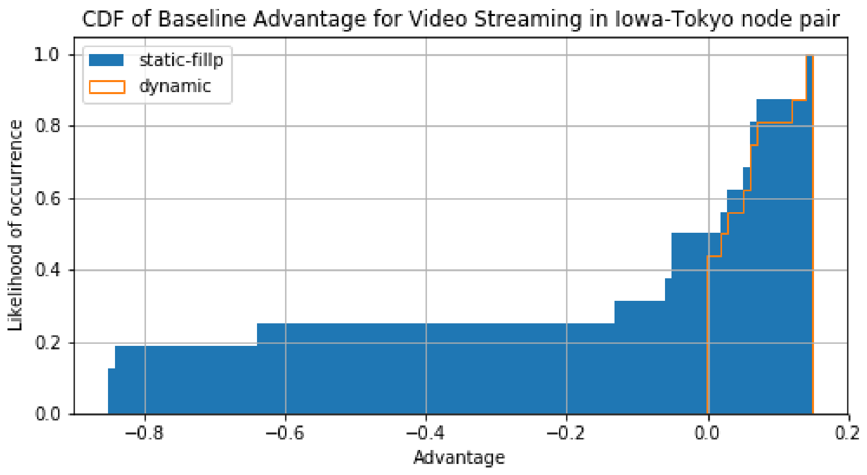


Figure 4.5: CDF for application utility in Iowa-Tokyo node pair for video streaming

Metric	Static Best	Percentage (%)
Throughput-only	FillP	100.0
Delay-and-throughput	FillP	93.75
Online Gaming	FillP	93.75

Table 4.7: Static best algorithms for different metrics in Tokyo-Sydney node pair.

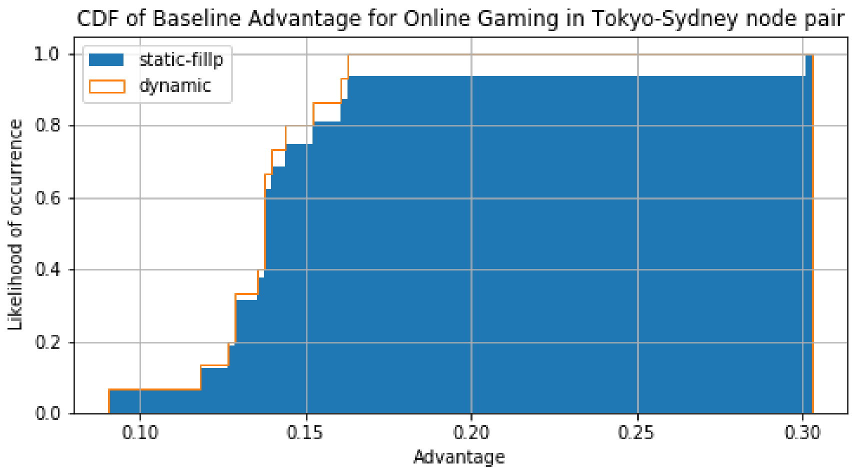


Figure 4.6: CDF for application utility in Tokyo-Sydney node pair for online gaming

rithm is TCP Cubic. The dynamic best algorithm shows advantage within range $[0, 0.2]$ for more than 90% measurements. For video streaming, the static best algorithm is FillP. For 20% measurements, FillP has shown loss more than -0.6. The dynamic best algorithm has advantage within 0.1 for all measurements. For online gaming, the static best algorithm is FillP. Surprisingly, it prevails TCP Cubic for all measurements. But it has advantage within 0.2 for 90% measurements. The dynamic best algorithm shows similar performance.

Sydney-London

There is an interesting phenomenon in this node pair. Figure 4.7 is shown as an example. For VoIP, the static best algorithm is Vivace. However, the performance of Vivace varies a lot across all measurements. Vivace shows advantage above 0.4 for 60% measurements. It means, for the majority, Vivace shows considerable gain against the default congestion control algorithm. Considering the gain from the best cases, it is helpful to switch to Vivace. But the loss for the worst cases cannot be negligible. Using dynamic best algorithm is a great choice in order to get rid of the loss. However, it is also noticed that the dynamic setting requires omniscient knowledge. It is hard to get such knowledge in real implementation.

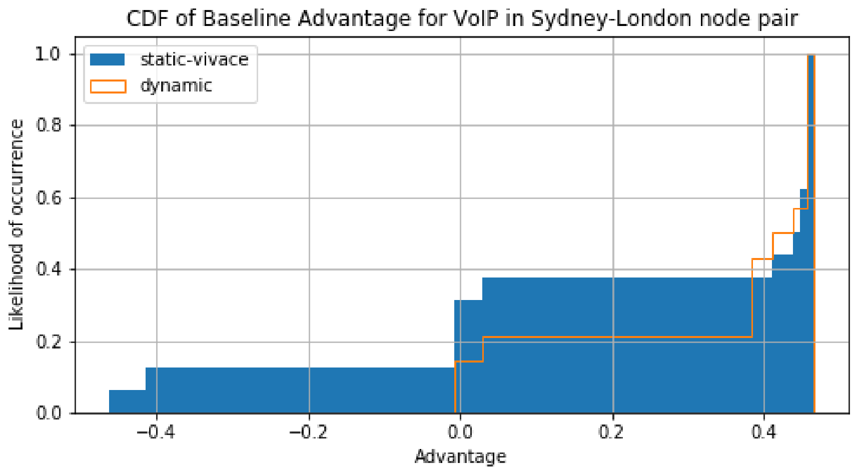


Figure 4.7: CDF for application utility in Sydney-London node pair for VoIP

4.2.4 Data flow direction

There are two kinds of data flow directions in Pantheon: download and upload. In this subsection, we would like to see if data flow direction affect congestion control algorithm performance and how many effects it has.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	28.13
Throughput-only	FillP	57.81
Loss-only	Vivace	42.19
Delay-and-throughput	Indigo	53.13

Table 4.8: Static Best algorithms for basic metrics in download.

Metric	Static Best	Percentage (%)
Delay-only	Vivace	42.19
Throughput-only	Indigo	42.19
Loss-only	Copa	18.75
Delay-and-throughput	FillP	29.69

Table 4.9: Static best algorithms for basic metrics in upload. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

First of all, there is no pattern for the choice of static best algorithm in both data flow direction. For each of the four basic metrics, the static best algorithm is chosen differently for download or upload (Table 4.8, 4.9). In other words, data flow direction does affect algorithm performance. However, the effect is on a case-by-case basis. It is hard to tell the pattern from Pantheon's data. It is also noticed that the percentage is no larger than 60% for all metrics presented in the tables. It means that the performance of algorithms varies across the measurements.

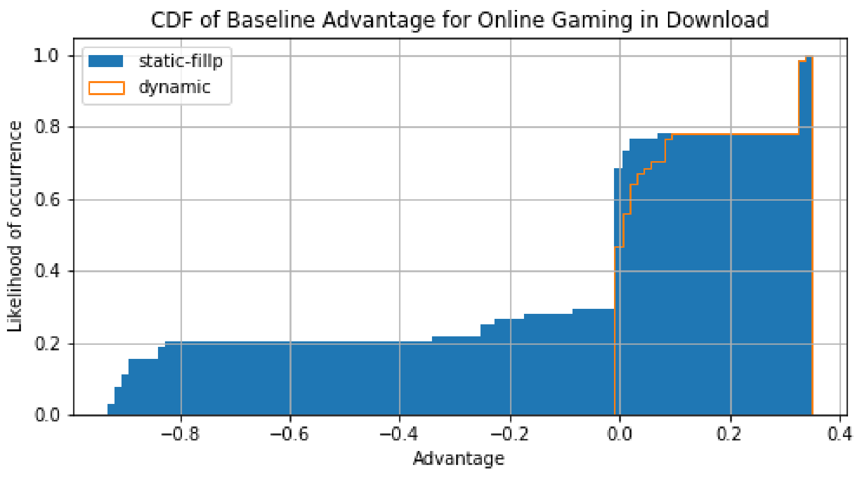


Figure 4.8: CDF for application utility in download for online gaming

In addition, it is not clear that meta congestion control benefits the application utility considerably. For instance, Figure 4.8 and Figure A.17 show CDF of online gaming application utility difference in download and upload. FillP is selected as the static best algorithm for both directions. In download direction, FillP's utility score is lower than the default algorithm TCP Cubic for more than 0.8 in 20% measurements. In about 80% measurements, FillP does not have advantage more than 0.2. Considering the worst cases, it may not be a good idea to switch from TCP Cubic to FillP. In upload direction, it is also noticed that the CDF has "a long tail", which indicated similar trend as the download direction shows. For the best measurements, FillP's advantage is below 0.1, which is not very promising. Hence, for both directions, meta congestion control has limited ability on increasing application utility.

4.2.5 Link type

Pantheon also conduct measurements on different link types. There are two kinds of link types, i.e. ethernet and cellular. Due to lack of data, we only use the measurement between Stanford and

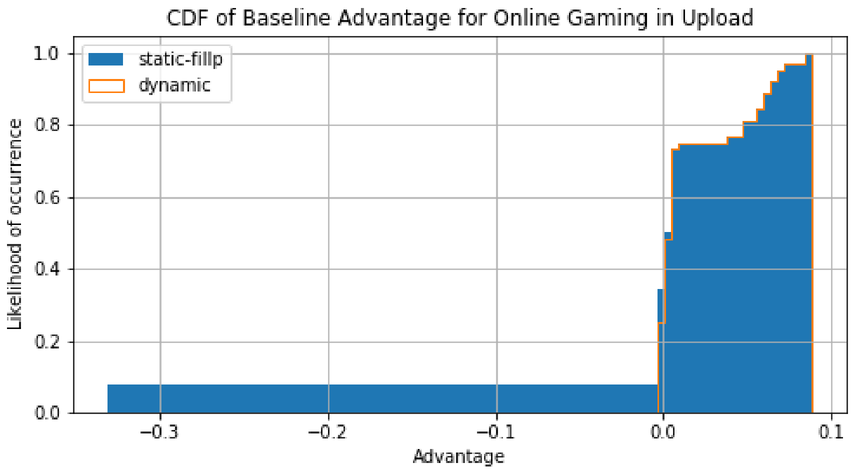


Figure 4.9: CDF for application utility in upload for online gaming

AWS California in the analysis.

Metric	Static Best	Percentage (%)
VoIP	BBR	75.00
Video Streaming	BBR	87.50
Online Gaming	BBR	81.25

Table 4.10: Static best algorithms for different metrics in ethernet.

Metric	Static Best	Percentage (%)
VoIP	Vegas	43.75
Video Streaming	Vegas	62.50
Online Gaming	Vegas	43.75

Table 4.11: Static best algorithms for different metrics in cellular.

Link type is a dominant factor on choosing static best algorithm for all three application scenarios. When the link type is ethernet, BBR is chosen as the static best algorithm (Table A.11). When the link type is changed to cellular, Vegas becomes the static best algorithm (Table A.12).

However, BBR does not show large advantage compared to the baseline algorithm TCP Cubic. For example, in Figure 4.11, although there are cases where BBR has loss compared to TCP Cubic, all loss is within 0.05. At the same time, there is no measurement where BBR has advantage over TCP Cubic above 0.1. Meta congestion control may not help user have much better utility over default congestion control algorithm here.

For cellular link type, it is noticed that the performance of congestion control algorithms differs a lot as the one we shown before in ethernet. For all three applications, the static best algorithm is Vegas. For example, in VoIP (Figure 4.11), Vegas has advantage above 0.6 for more than 50% measurements. The dynamic best algorithm performs even better, its advantage is in range $[0.6, 1]$ for all measurements. The other two applications have similar results (Figure A.19). In this

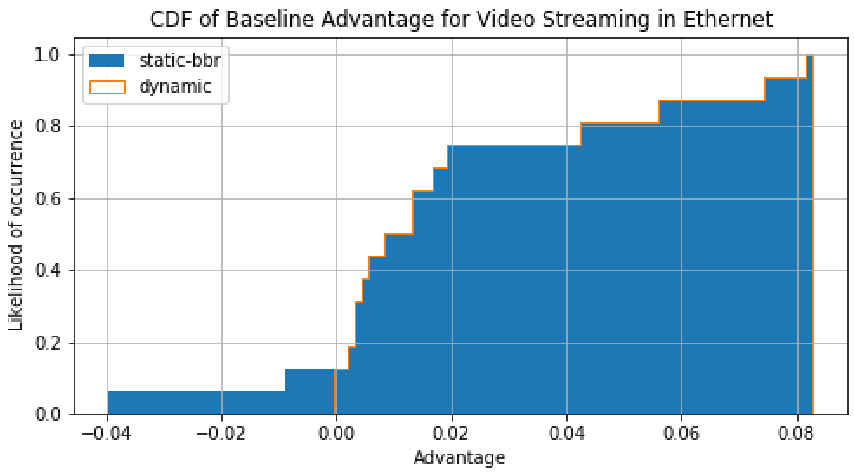


Figure 4.10

Figure 4.11: CDF for application utility in ethernet for video streaming

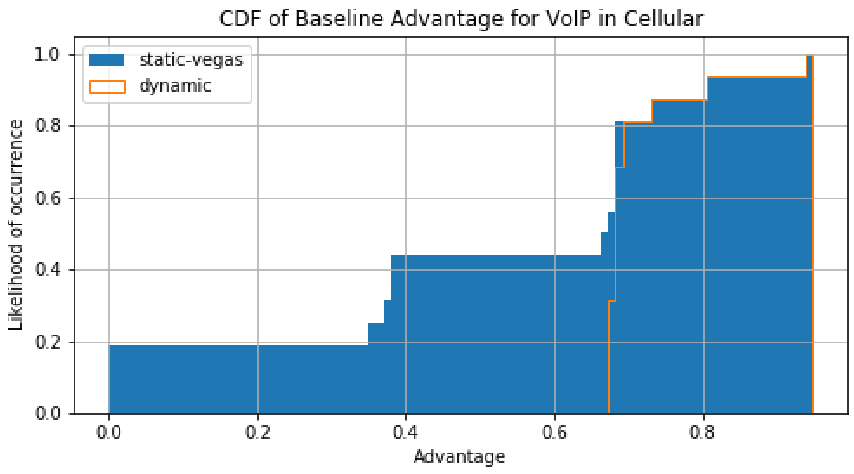


Figure 4.12

Figure 4.13: CDF for application utility in cellular for VoIP

case, switching from TCP Cubic to Vegas can increase application utility considerably for most measurements.

Chapter 5

Conclusion and Discussion

This chapter concludes the results which are obtained from previous analysis and develop further discussions. In former chapter, I investigate congestion control algorithm performance under different criteria. With those quantifiable results in hand, it would be beneficial to conclude from a higher level to understand what is the takeaway from parallel evaluation sections. In the second part of this chapter, I discuss the limits in our conclusion and possibilities for future work.

5.1 Conclusion

There is no "single best" congestion control algorithm. On the one hand, there are different needs for various scenarios. It leads to different "best" congestion control algorithms. Data analysis shows that algorithm performance ranking varies according to the metric it uses. Even if it only considers three basic measurements, i.e. delay, throughput, loss, the congestion control with best performance could be different. Therefore, it would be beneficial to consider the specific need in the case where it will use these congestion control algorithms in order to leverage network resource. On the other hand, if the evaluation metric is fixed, i.e. the need is defined, the best algorithm could still vary according to node location, data flow direction, link time, time... Hence, congestion control algorithm should be implemented accordingly with, not just the need, but all the factors in mind.

Source node location does not considerably affect QUIC Cubic's low-delay character. Actually, for nodes in London, Iowa, Tokyo, QUIC Cubic has the lowest delay in more than 90% measurements. QUIC is a transport layer network protocol which is designed to be nearly equivalent to Transmission Control Protocol (TCP). However, it aims to has much-reduced latency. This could be the reason why QUIC Cubic shows great performance on delay for all four source nodes. Notice that all four nodes are "cloud-to-cloud" tests. In "node-to-cloud" test, QUIC Cubic does not have dominant performance on delay, which it is seen in other evaluation.

There is no obvious pattern on source node location for choosing the "best" congestion control algorithm for different application scenarios. The choice of algorithm should be on a case-by-case basis. When the application changes, the static best algorithm which has the best utility for the most measurements also changes (even though the source node remains the same). There is no single static best algorithm that has the best utility for all three application in one source node.

Destination node location is an important factor on algorithm performance. There is no significant benefit of algorithm switching for application utility. It is shown that the choice of static best algorithm varies according to destination node location. There are two destination nodes in Pantheon: California and Brazil. In California node, the worst cases of static best algorithm refrain from switching to it. For most measurements (80%), the dynamic best algorithm in omniscient setting does not increase the utility a lot. In Brazil node, TCP Cubic itself is selected as the static best algorithm for applications. For all three application, the dynamic best algorithm does not have large advantage (larger than 0.2) over TCP Cubic for a lot of measurements (60%).

The performance of meta congestion control varies a lot for each node pair. It does not help a lot on improving utility in most node pairs for different reasons. First, the static best algorithm, or even dynamic best algorithm with omniscient view, has only slight advantage over TCP Cubic. In addition, there are cases where the loss of using algorithm other than TCP Cubic is large. In other words, meta congestion control does not show considerable advantage over TCP Cubic in most measurements while it suffers from utility loss in the worst cases.

Meta congestion control with omniscient setting does not increase application utility for ethernet link. However, switching congestion control algorithm can increase application utility for cellular link. For ethernet link, even the omniscient setting does not show significant increase on application utility. On the contrary, switching from TCP Cubic baseline algorithm to Vegas will increase the utility for most measurements. If the dynamic switching is available, application utility will increase even further. Therefore, implementing meta congestion control can benefit application usage in cellular link.

There are cases where FillP prevails the others for one-year-long measurements. More design details on FillP are needed in order to explain its performance. Especially, FillP is ranked in the first place for 100% measurements in throughput-only metric in Table 4.6 and Table 4.7. It is surprising that FillP can have the highest throughput for such a long time range. However, the reason why FillP has such prominent performance is not clear. It is an "ongoing" congestion control algorithm according to Pantheon website. Therefore, it is hardly known to us why it could have such performance without revealing its design in more detail.

5.2 Discussion

Given all takeaways shown above, there are several points that worth discussing.

Although Pantheon is seen as the state-of-the-art platform in congestion control, it has only 16 nodes with network paths between them. Pantheon does not have nearly the scale of a commercial website in real world. Hence, it is possible that there are missing parts when this thesis only uses data from this platform. Especially, when we consider that the diversity of large-scale network leads to the complexity on choosing best congestion control algorithm, it remains unclear that how confident we should be for the conclusions. However, Pantheon is larger and more comprehensive than most academic congestion control evaluations. And we do see diversities in the results even though the data analysis is based on these 16 notes. At least, it gives us an idea on the difficulty of implementing meta congestion control and the potential of limited advantage.

Application utility analysis is an important part in this thesis. The goal is to see whether meta congestion control can benefit for specific application usage. The utility function and utility score is developed for each application respectively. However, the test traffic of Pantheon is consistent for all measurements. It is a full-throttle flow. The idea is to implement a least-common-denominator for all different congestion control schemes. It means that the workload itself is not customized for application. There is an underlying assumption in the data analysis that the performance of algorithms should not vary a lot for different workload patterns. However, more work is needed to validate this assumption.

Chapter 6

Summary

This thesis evaluates the idea of meta congestion control by analyzing data from Pantheon. It is shown that there is no one-single algorithm that prevails all the other ones in all network conditions and application scenarios. Therefore, switching congestion control algorithm is an intuitive way to leverage network resource for usage situation. However, it also shows that there is no obvious pattern for which algorithm performs the best in some scenarios. The lack of pattern becomes an obstacle of implementing meta congestion control since the meta protocol hardly knows which algorithm it should switch to with limited-offline knowledge of networks. The optimal algorithm should be selected on a case-by-case basis. In addition, the potential gain of meta congestion control varies in usage scenarios. While few cases show promising increase on application utility, others do not. It draws shadow on implementing meta congestion control in a large scale.

Bibliography

[1] Arun, V., and Balakrishnan, H. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (Renton, WA, Apr. 2018), USENIX Association, pp. 329–342.

[2] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. Tcp vegas: New techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev. 24*, 4 (Oct. 1994), 24–35.

[3] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. Bbr: Congestion-based congestion control. *ACM Queue 14, September-October* (2016), 20 – 53.

[4] Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M. PCC: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 395–408.

[5] Dong, M., Meng, T., Zarchy, D., Arslan, E., Gilad, Y., Godfrey, B., and Schapira, M. PCC vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (Renton, WA, Apr. 2018), USENIX Association, pp. 343–356.

[6] Floyd, S. Highspeed tcp for large congestion windows. RFC 3649, RFC Editor, 12 2003.

[7] Gerla, M., Sanadidi, M. Y., Ren Wang, Zanella, A., Casetti, C., and Mascolo, S. Tcp westwood: congestion window control using bandwidth estimation. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)* (2001), vol. 3, pp. 1698–1702 vol.3.

[8] Ha, S., Rhee, I., and Xu, L. Cubic: a new tcp-friendly high-speed tcp variant. *Operating Systems Review 42* (07 2008), 64–74.

[9] Henderson, T., Floyd, S.and Gurtov, A., and Nishida, Y. The newreno modification to tcp's fast recovery algorithm, rfc 6582, doi 10.17487/rfc6582, 2012.

[10] Kurose, J. F., and Ross, K. W. *Computer Networking: A Top-Down Approach (6th Edition)*. 2007.

[11] McMillan, K., and Zuck, L. Formal specification and testing of quic. pp. 227–240.

[12] Mohanty, S. R., Liu, C., Liu, B., and Bhuyan, L. N. Max-min utility fairness in link aggregated systems. In *2007 Workshop on High Performance Switching and Routing* (2007), pp. 1–7.

[13] Mu, M., Mauthe, A., and Garcia, F. A utility-based qos model for emerging multimedia applications.

[14] Nagaraj, K., Chinchali, S., Bharadia, D., Mao, H., Attar, M., and Katti, S. Numfabric: Fast and flexible bandwidth allocation in datacenters.

[15] Turkovic, B., Kuipers, F. A., and Uhlig, S. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv e-prints* (Mar. 2019), arXiv:1903.03852.

[16] Wikipedia contributors. Quic — Wikipedia, the free encyclopedia, 2020. [Online; accessed 8-June-2020].

[17] Yan, F. Y., Ma, J., Hill, G. D., Raghavan, D., Wahby, R. S., Levis, P., and Winstein, K. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 731–743.

[18] Zaki, Y., Pötsch, T., Chen, J., Subramanian, L., and Görg, C. Adaptive congestion control for unpredictable cellular networks. *ACM SIGCOMM Computer Communication Review 45* (08 2015), 509–522.

Appendix A

Plots and Tables

A.1 Plots

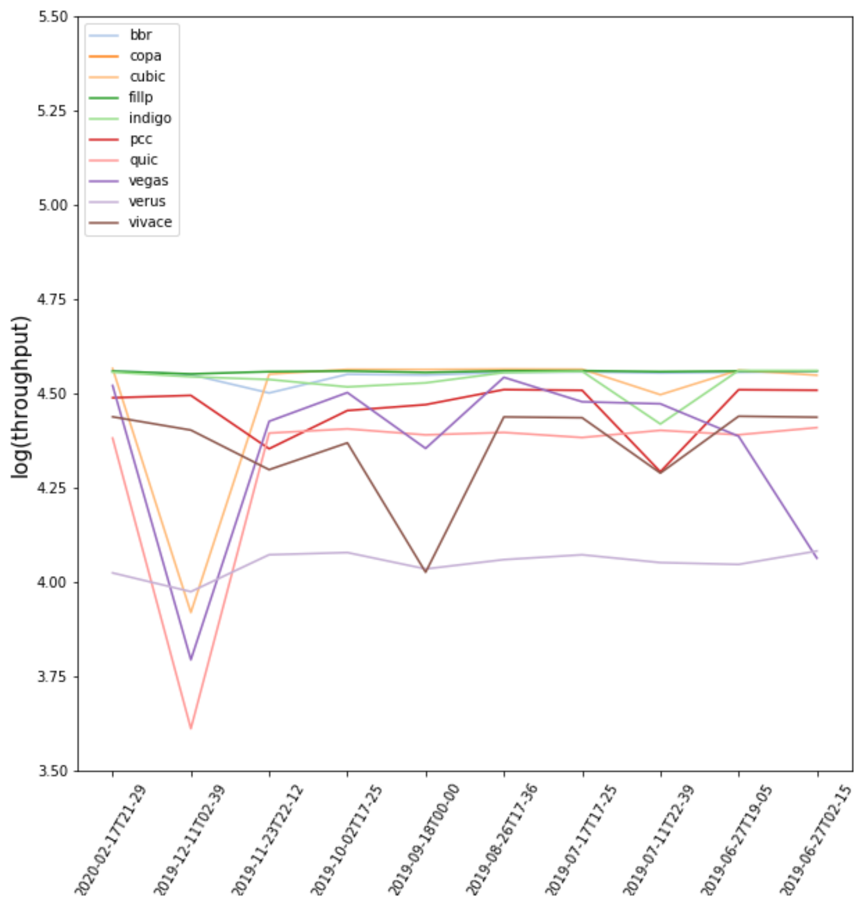


Figure A.1: Congestion control algorithms performance using throughput-only metric. Data are from a test node in Mexico.

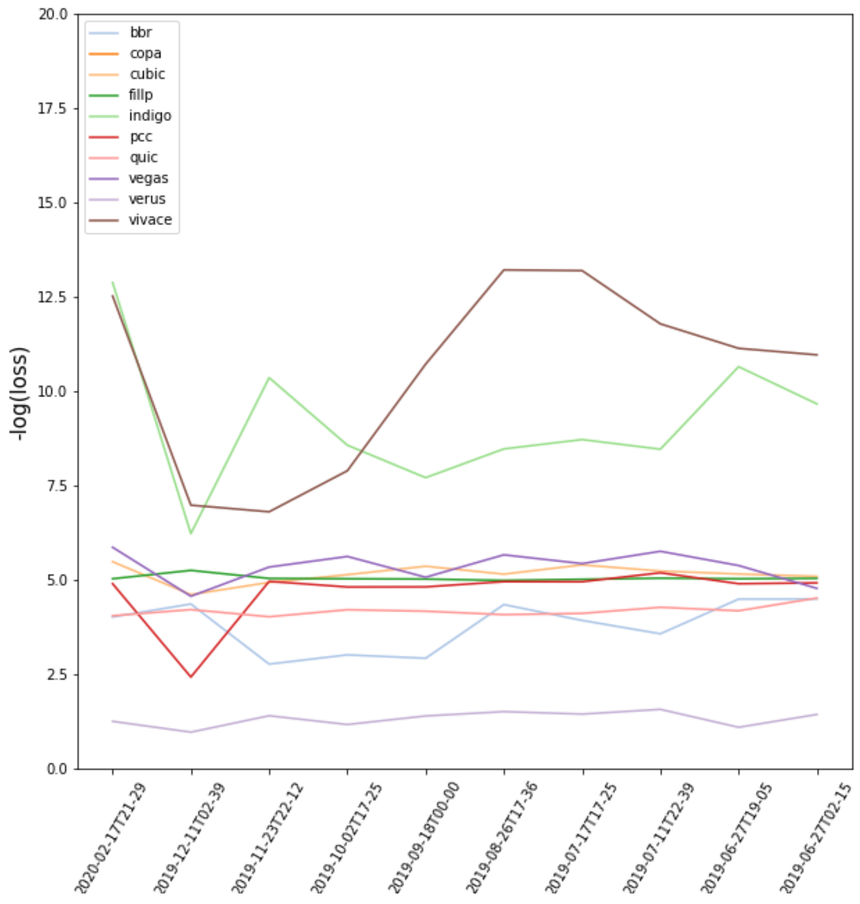


Figure A.2: Congestion control algorithms performance using loss-only metric. Data are from a test node in Mexico.

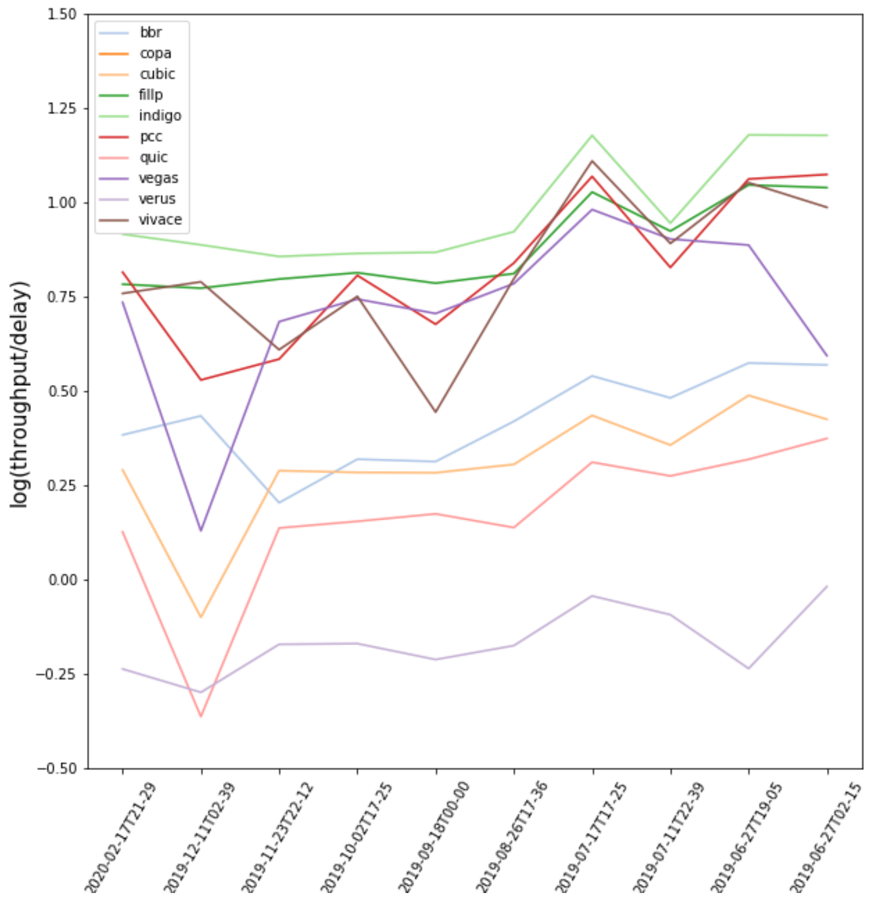
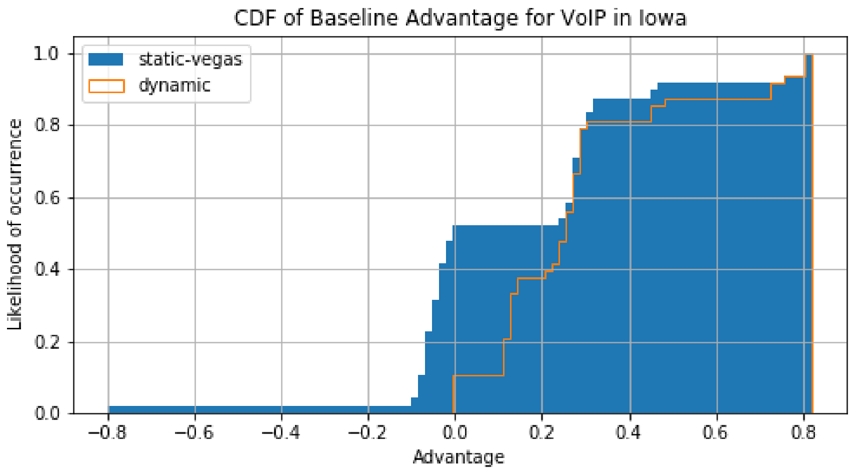
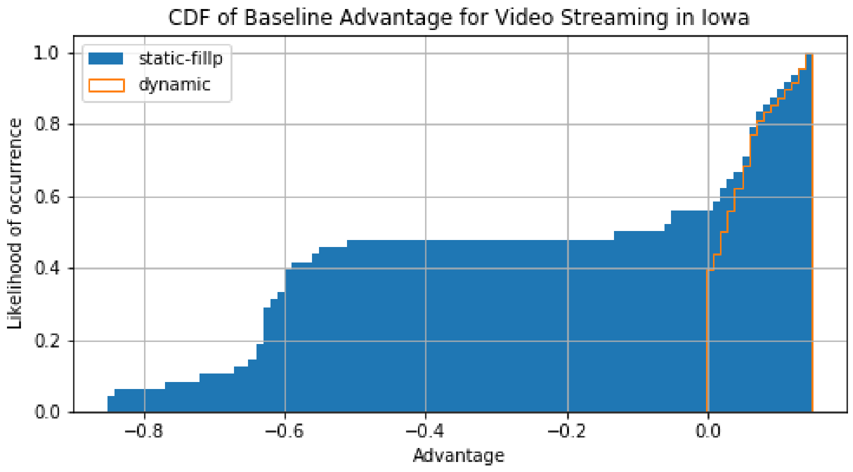


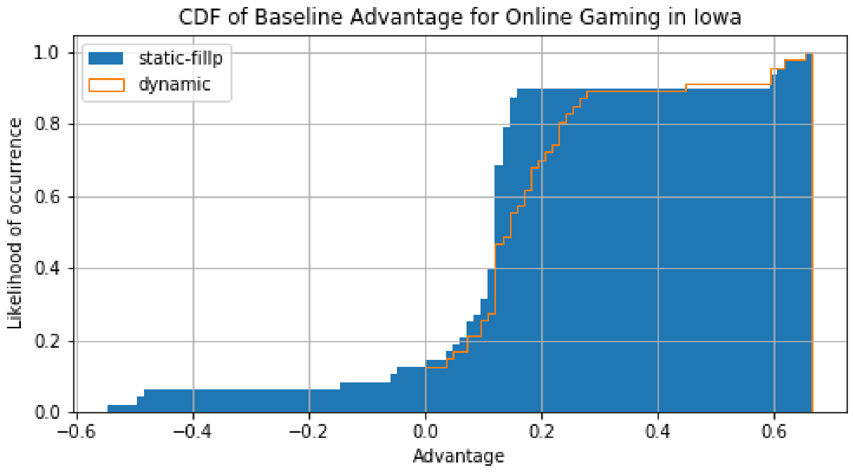
Figure A.3: Congestion control algorithms performance using both throughput and delay. Data are from a test node in Mexico.



(a)

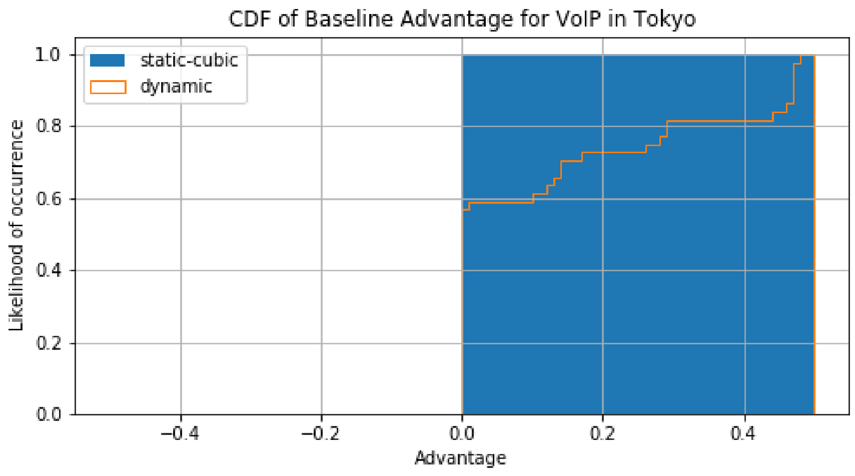


(b)

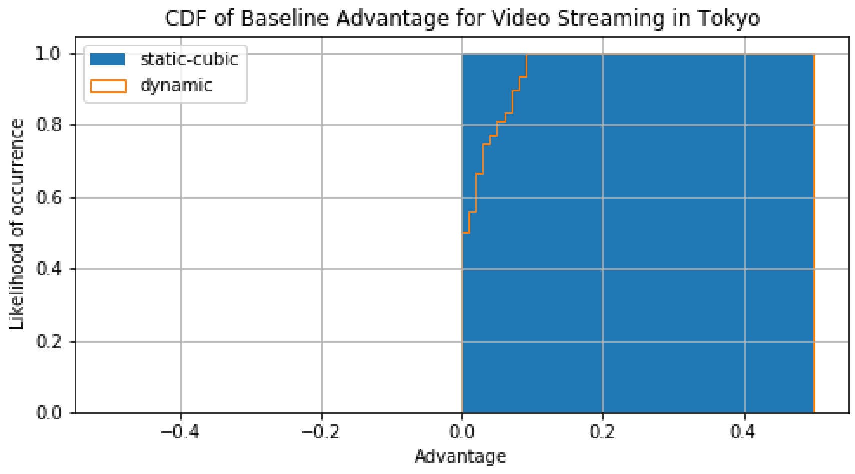


(c)

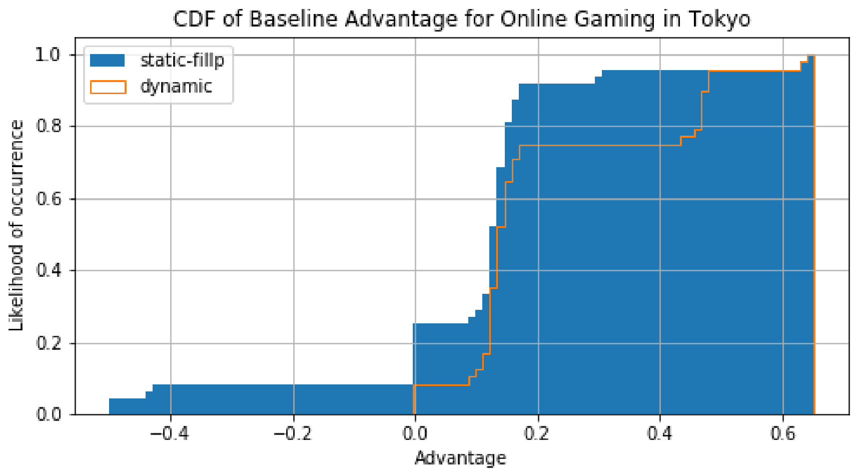
Figure A.4: CDF for application utility in Iowa



(a)

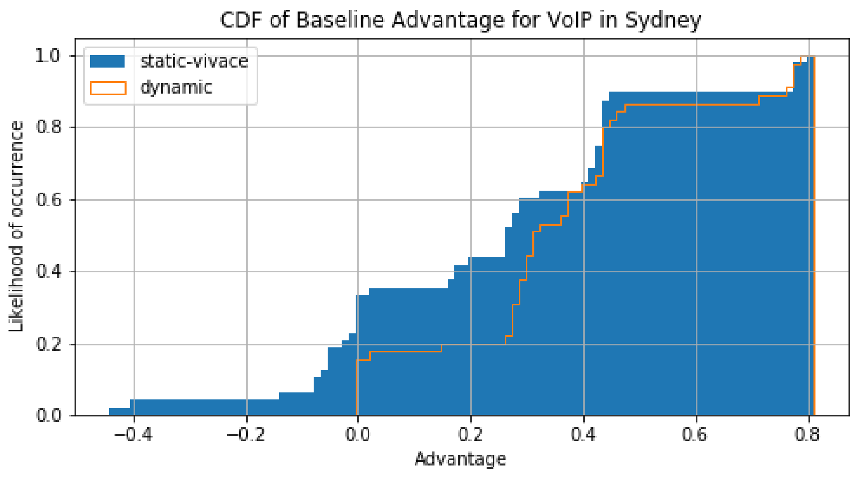


(b)

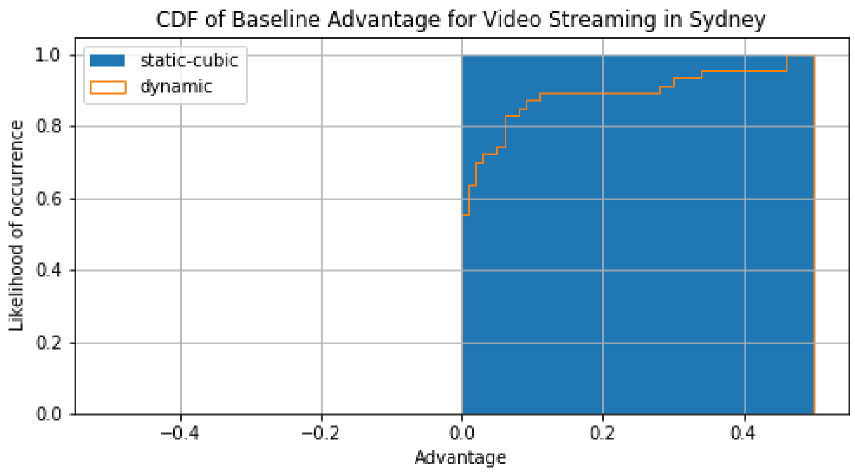


(c)

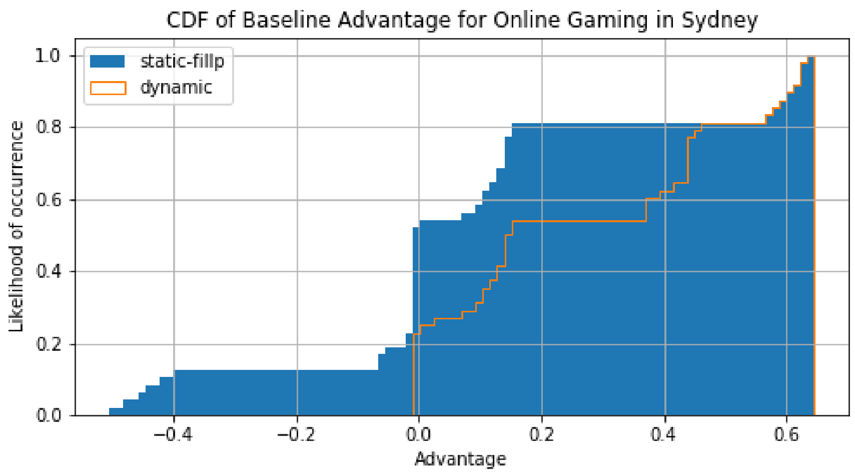
Figure A.5: CDF for application utility in Tokyo



(a)

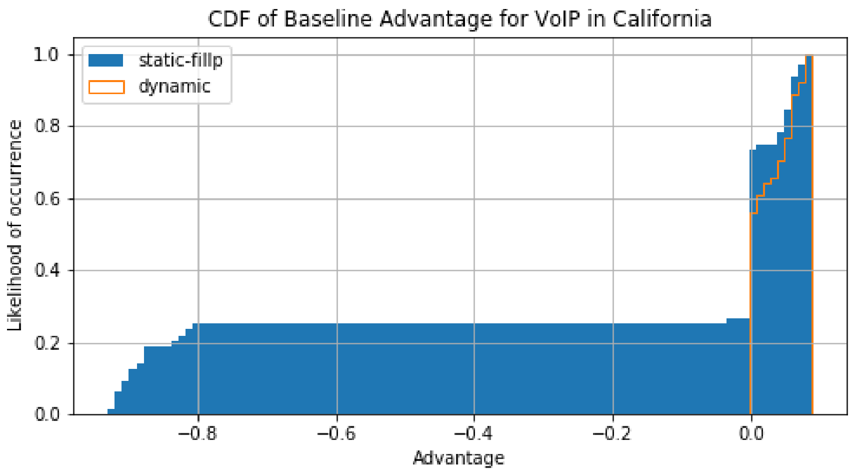


(b)

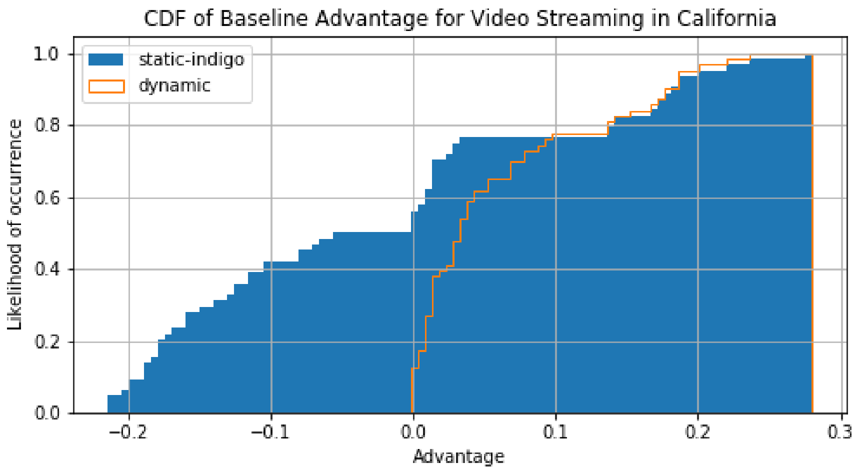


(c)

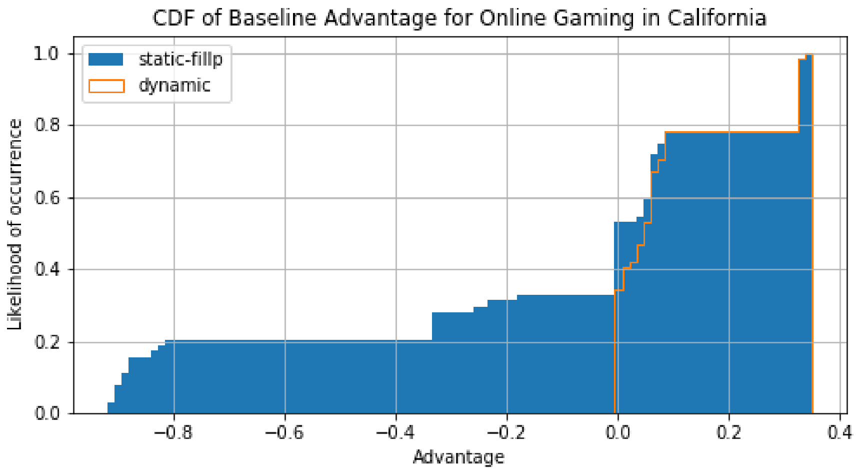
Figure A.6: CDF for application utility in Sydney



(a)

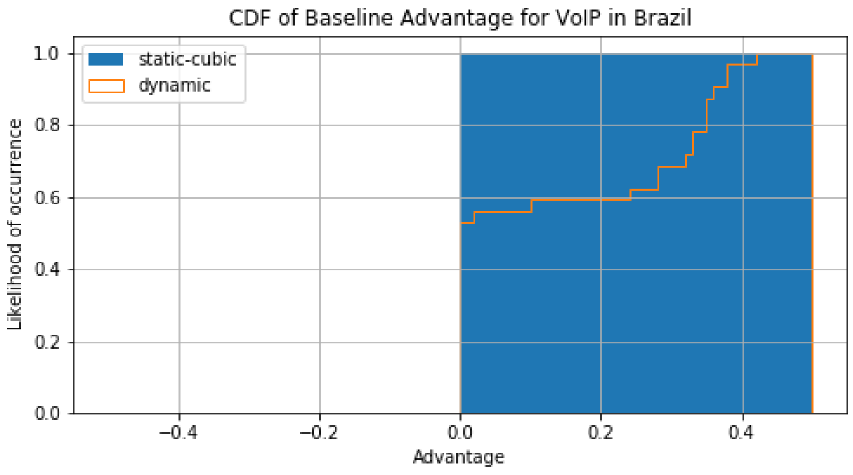


(b)

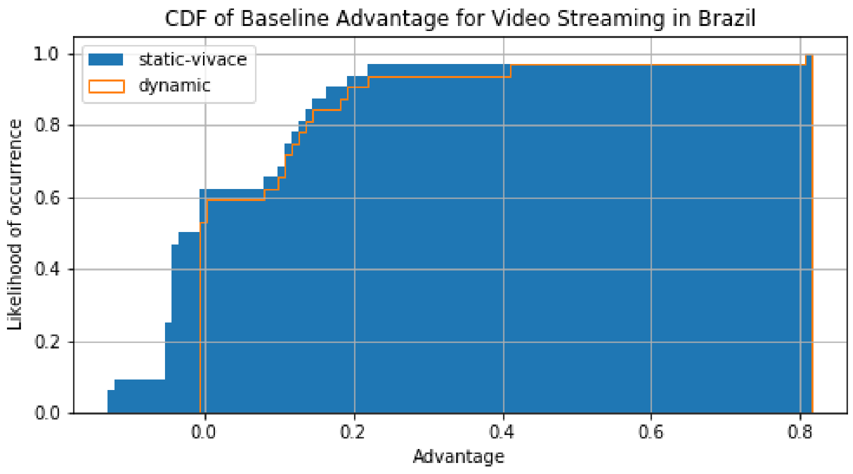


(c)

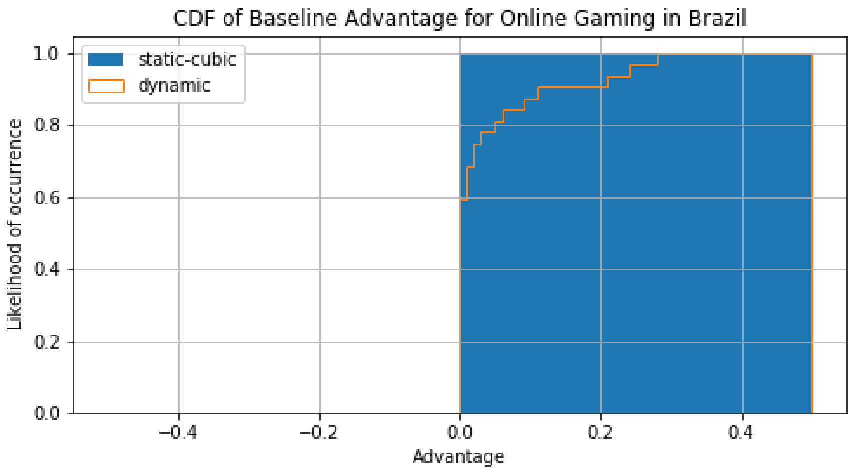
Figure A.7: CDF for application utility in California



(a)

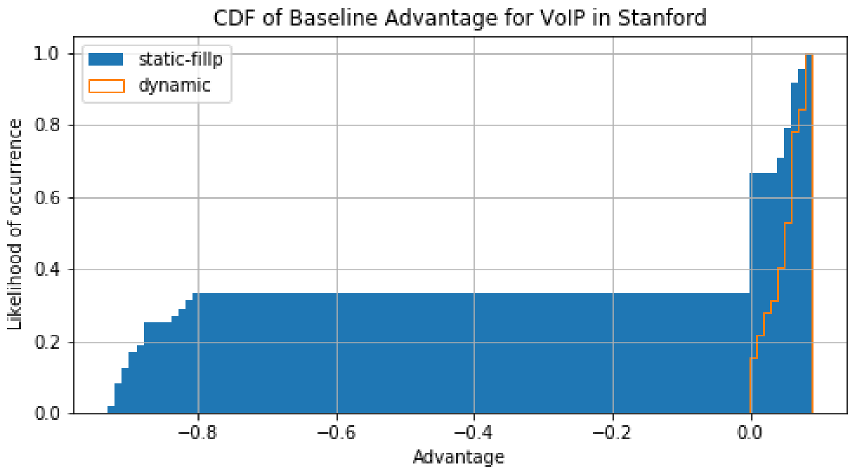


(b)

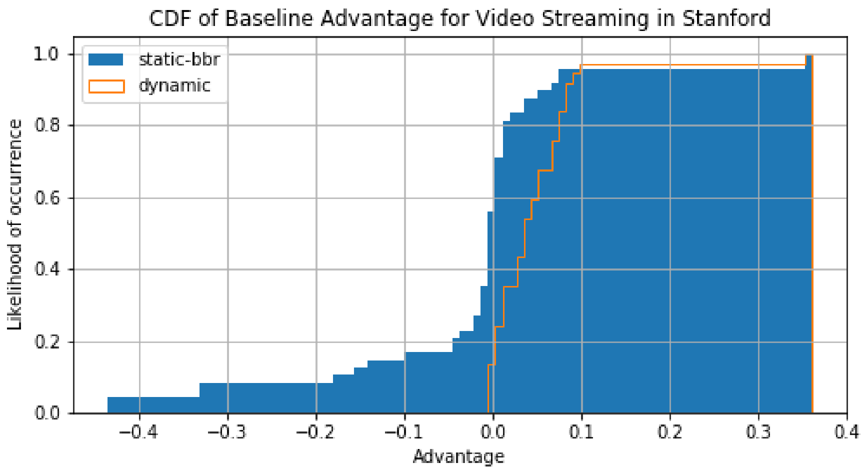


(c)

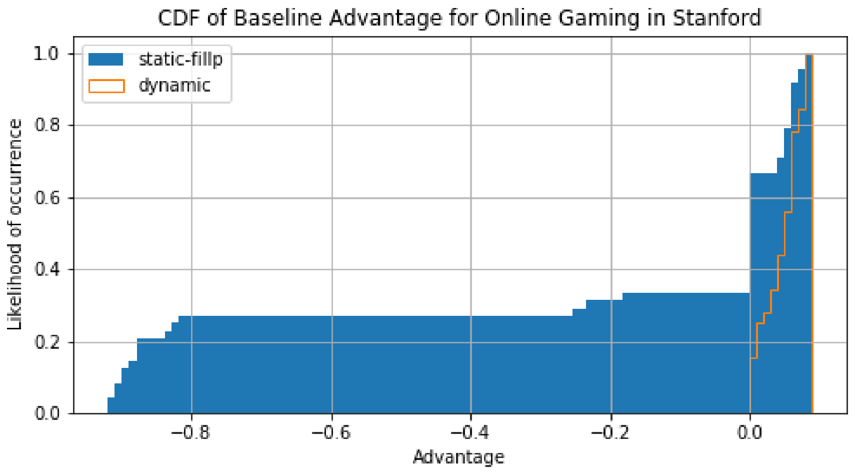
Figure A.8: CDF for application utility in Brazil



(a)

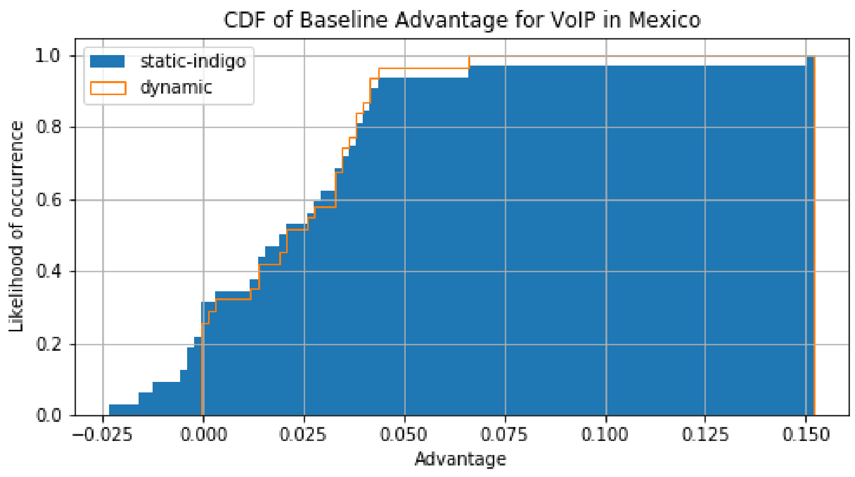


(b)

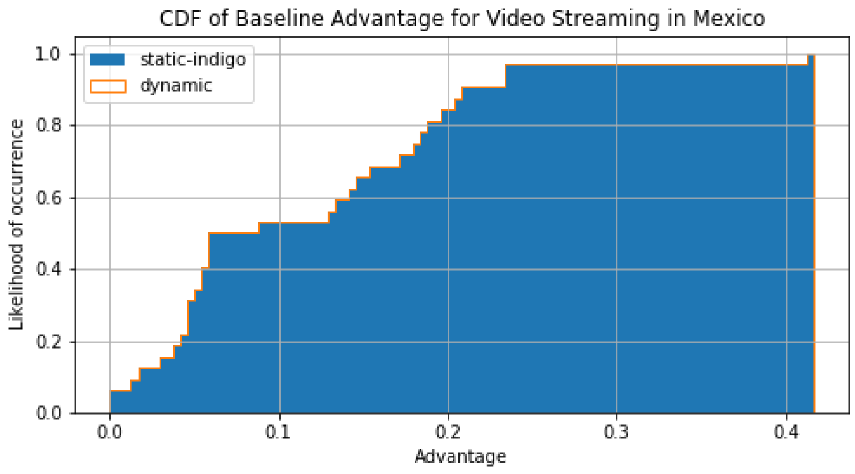


(c)

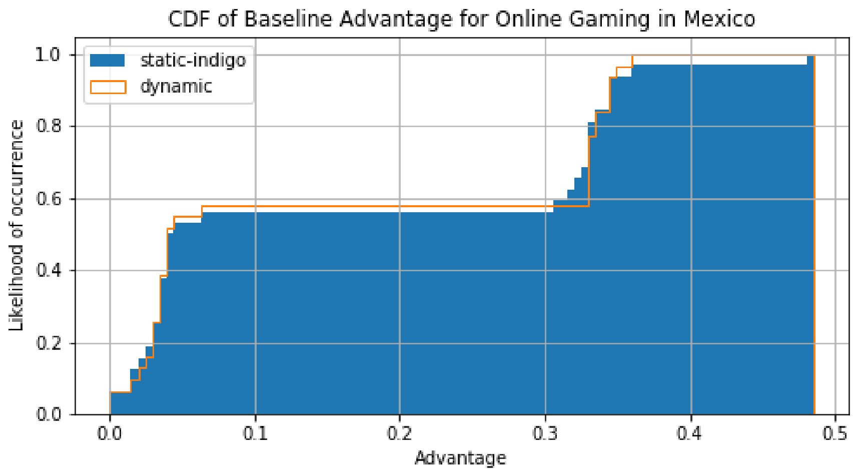
Figure A.9: CDF for application utility in Stanford-AWS California node pair



(a)

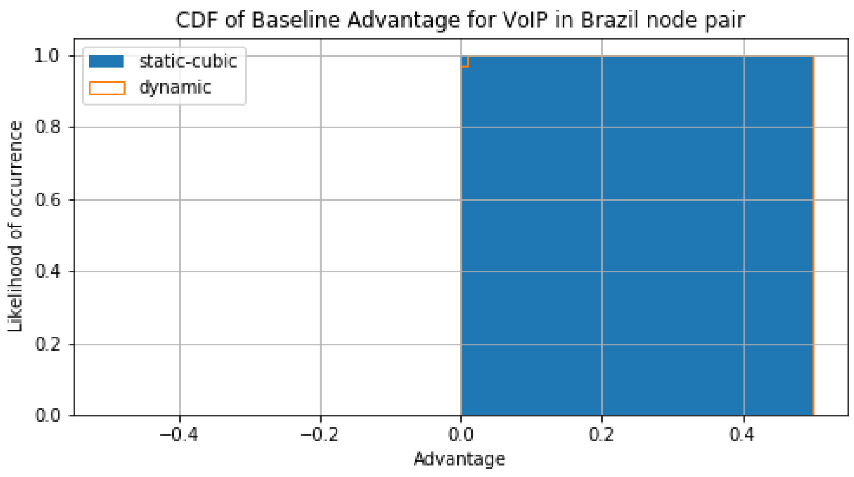


(b)

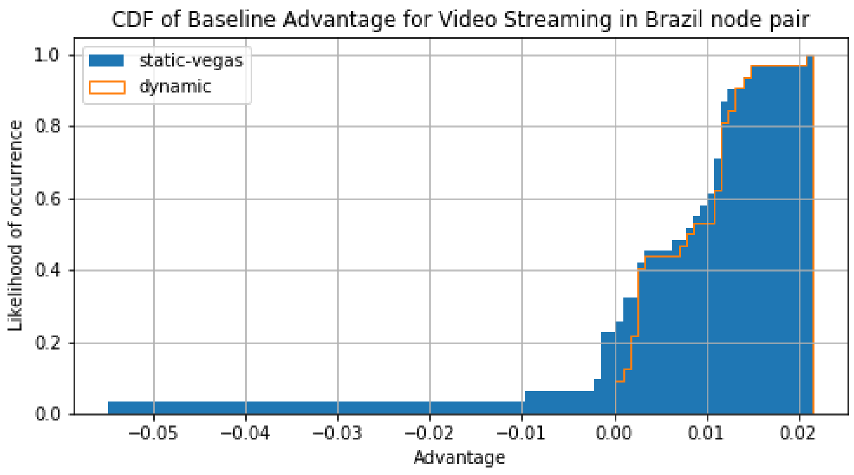


(c)

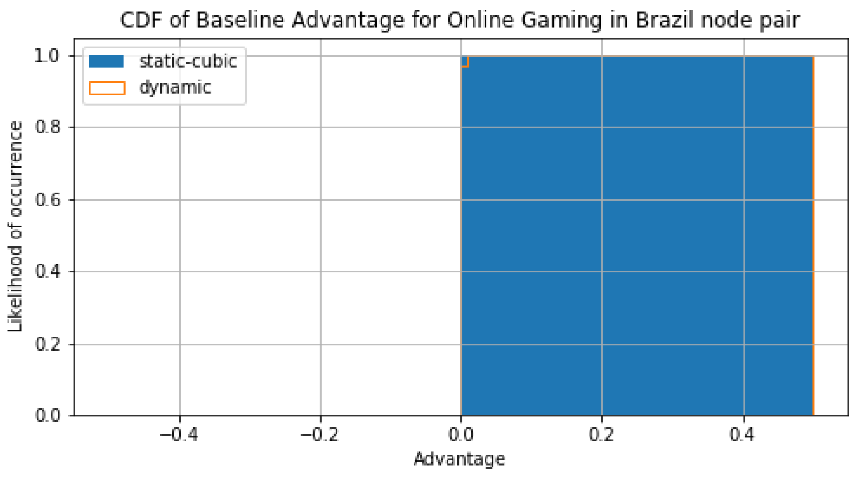
Figure A.10: CDF for application utility in Mexico node pair



(a)

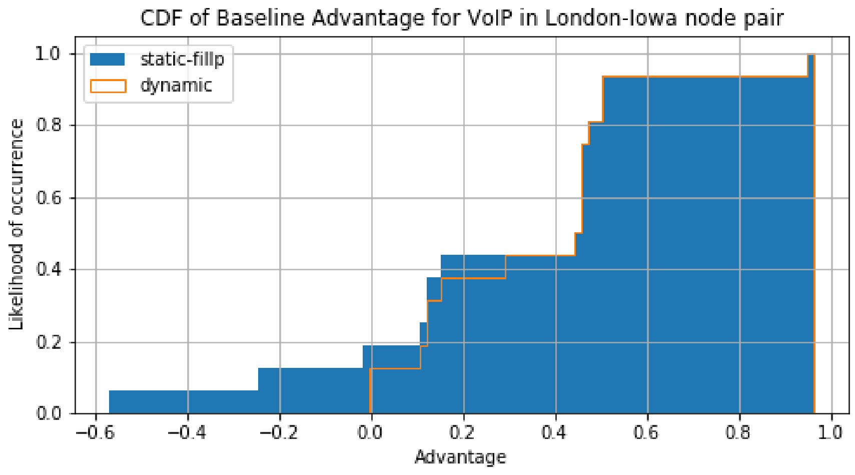


(b)

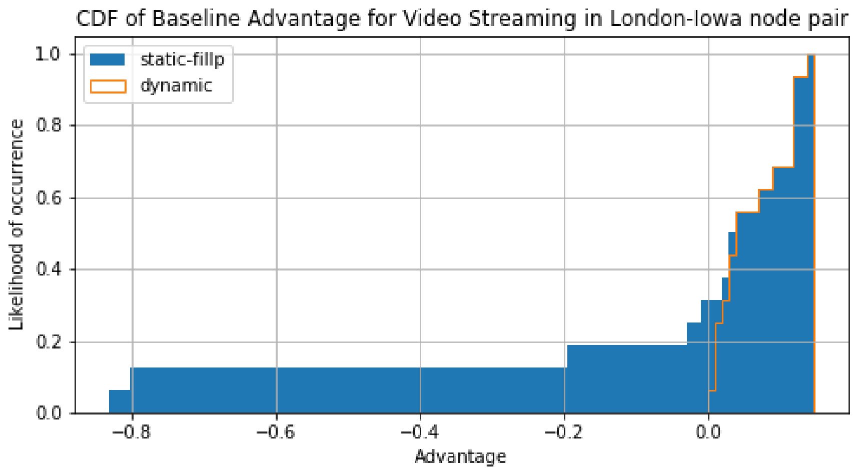


(c)

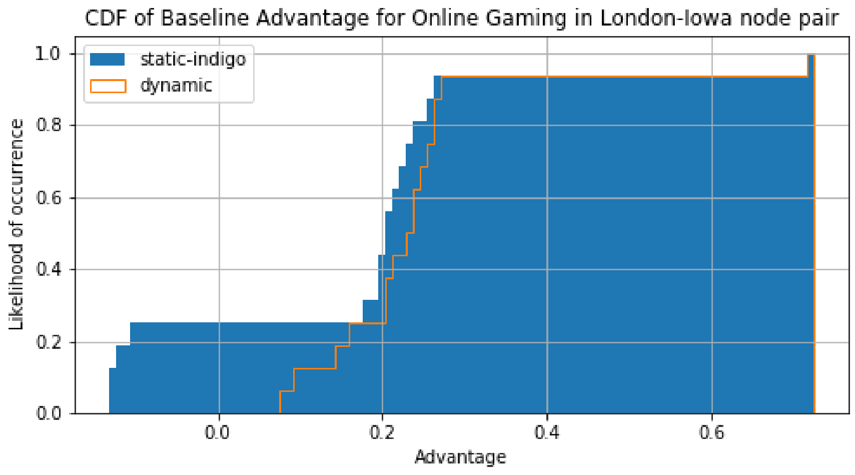
Figure A.11: CDF for application utility in Brazil node pair



(a)

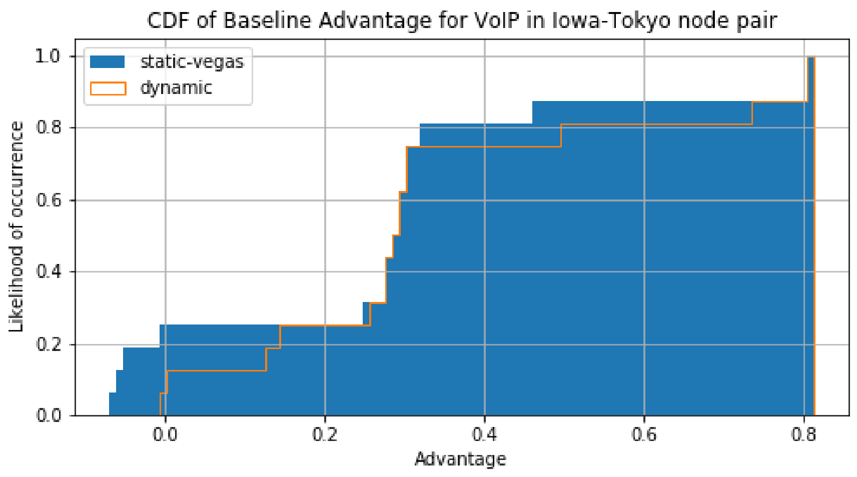


(b)

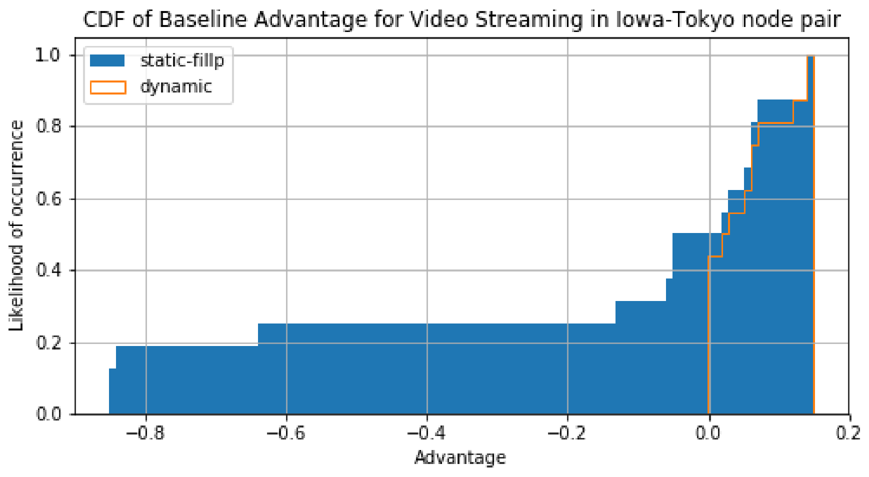


(c)

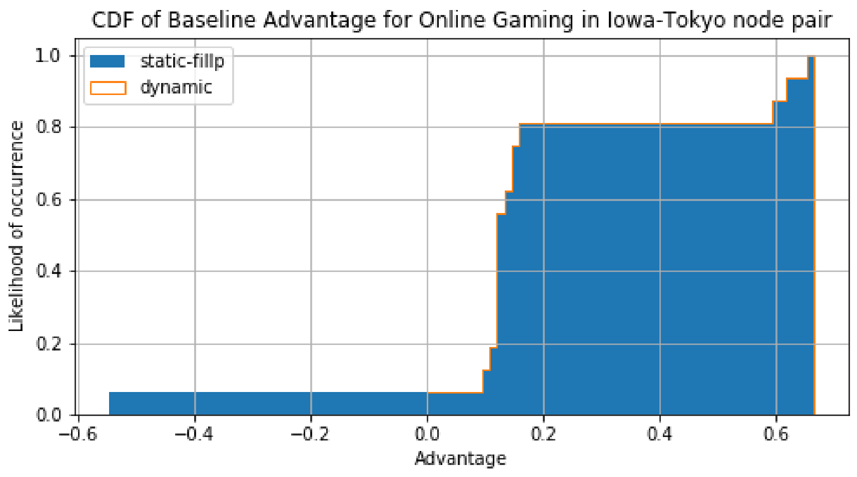
Figure A.12: CDF for application utility in London-Iowa node pair



(a)

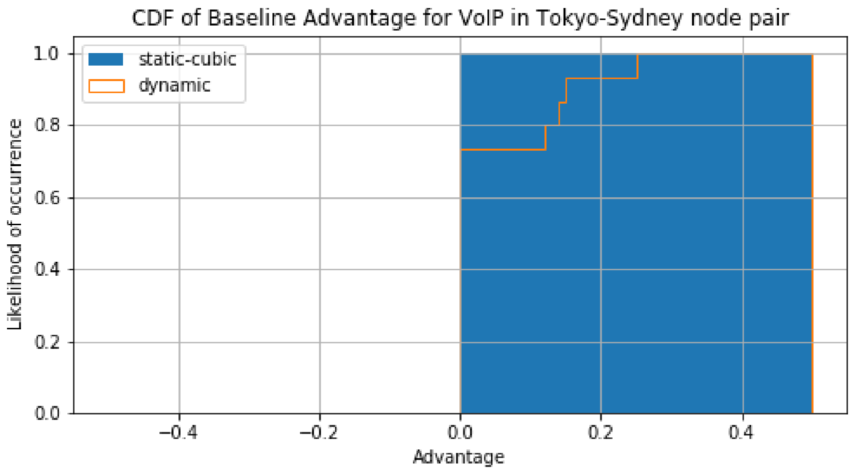


(b)

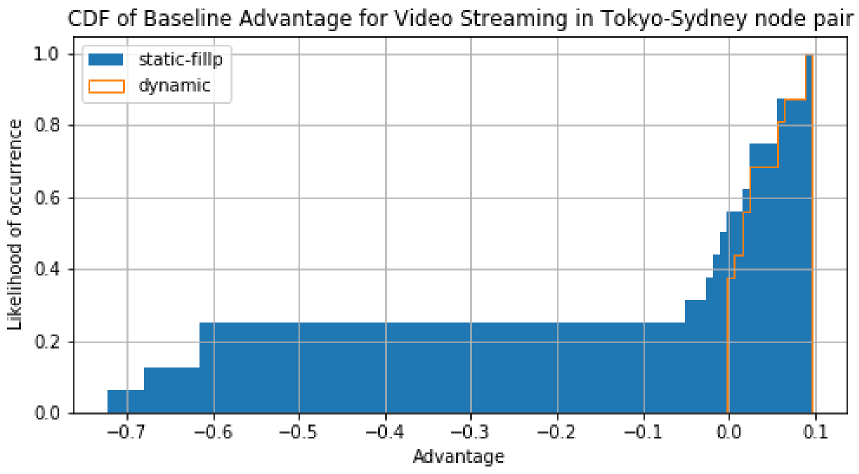


(c)

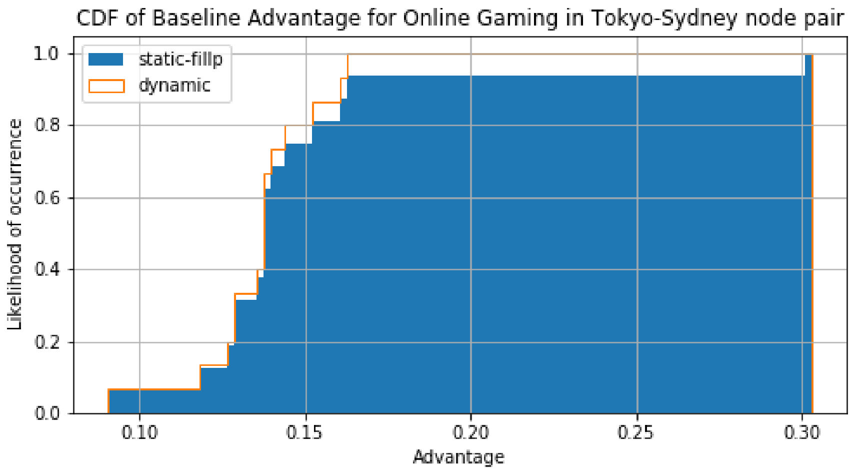
Figure A.13: CDF for application utility in Iowa-Tokyo node pair



(a)

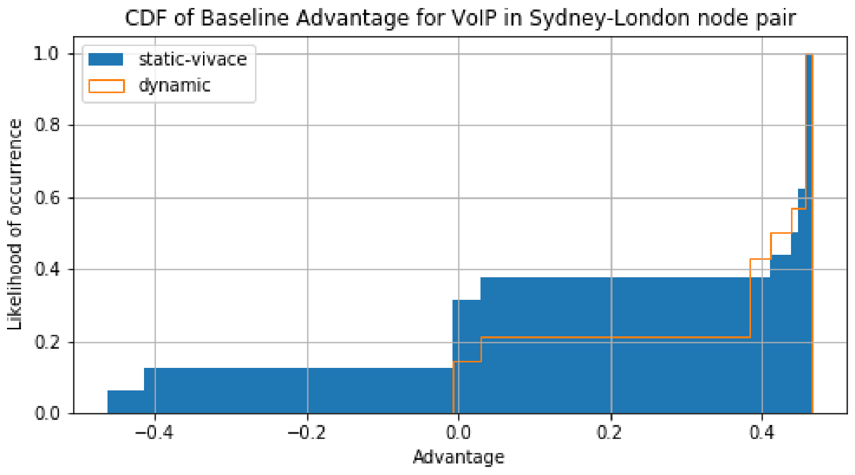


(b)

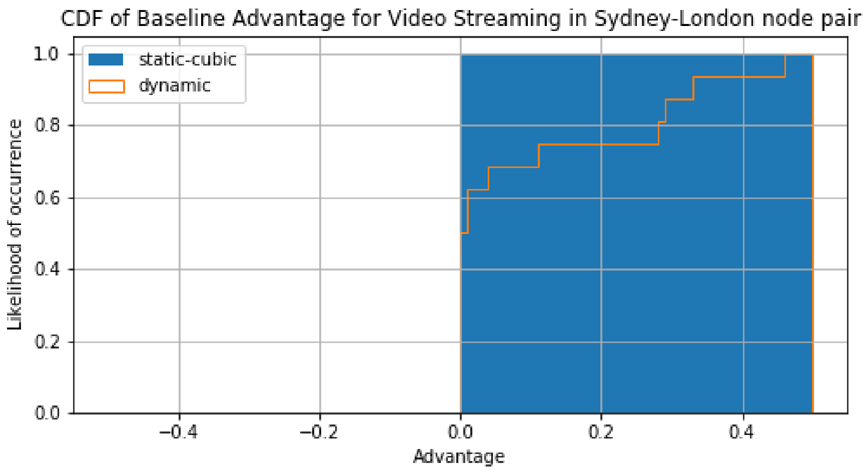


(c)

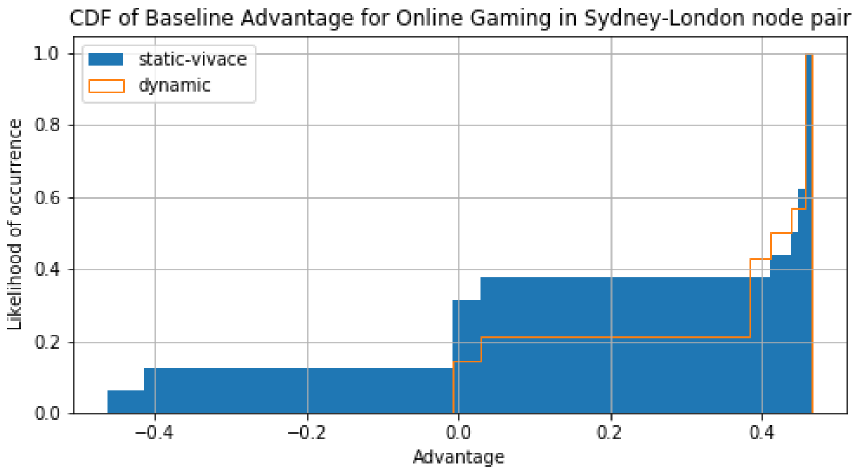
Figure A.14: CDF for application utility in Tokyo-Sydney node pair



(a)

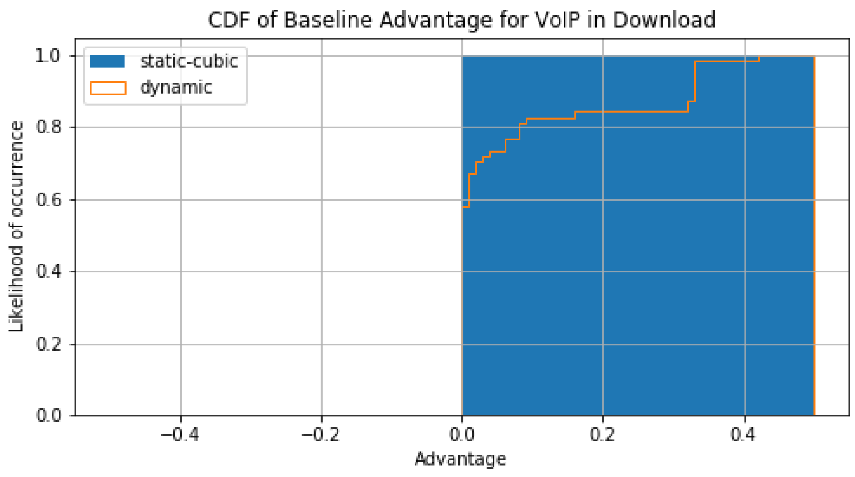


(b)

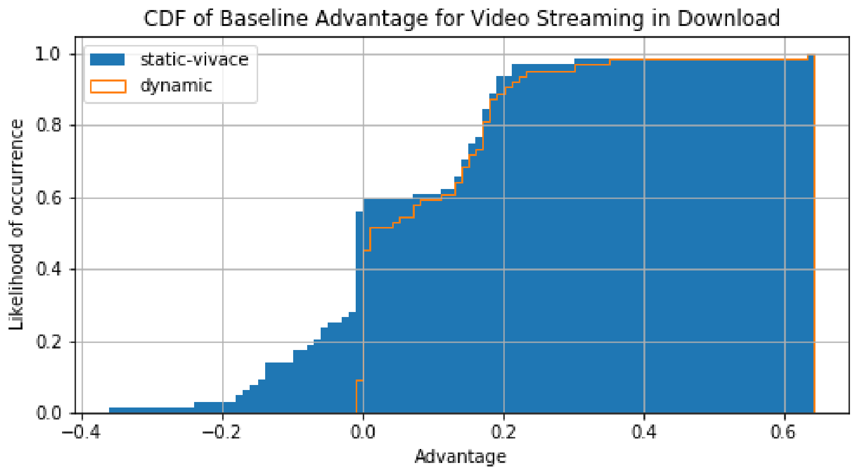


(c)

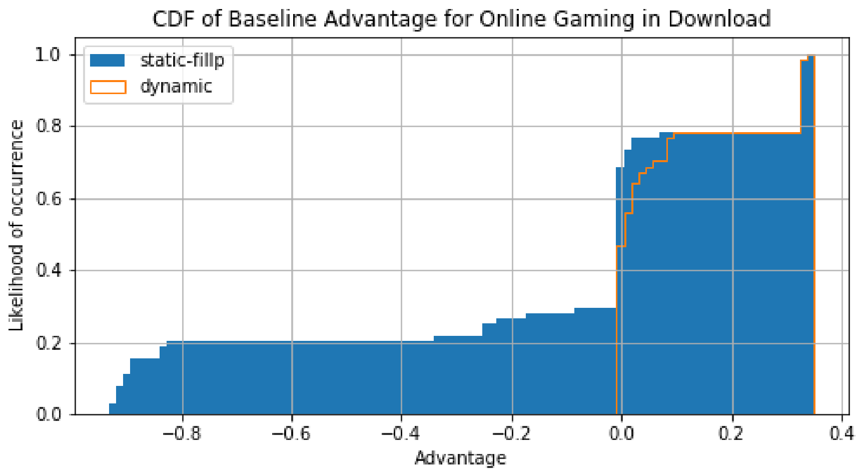
Figure A.15: CDF for application utility in Sydney-London node pair



(a)

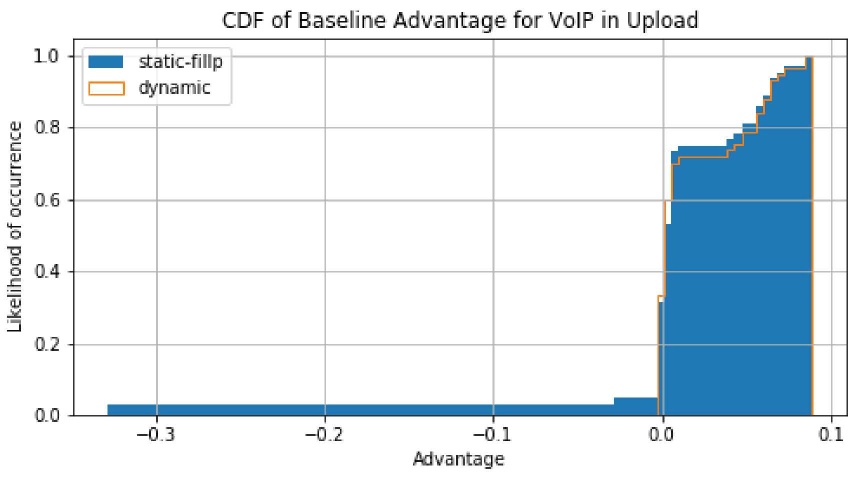


(b)

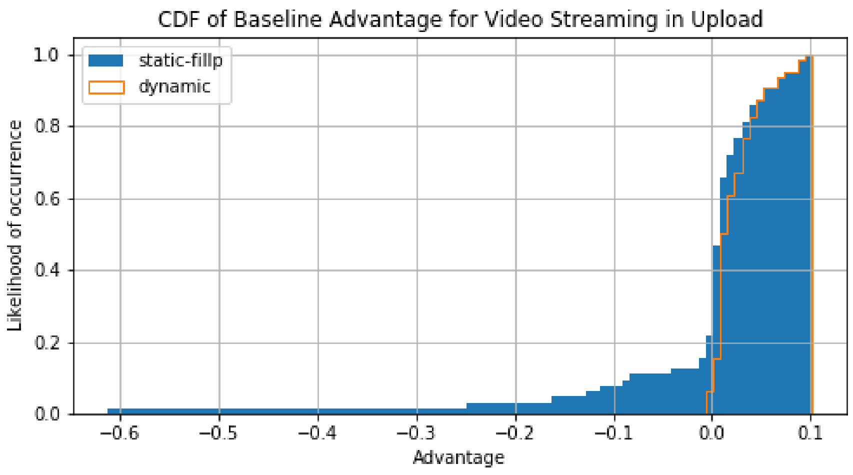


(c)

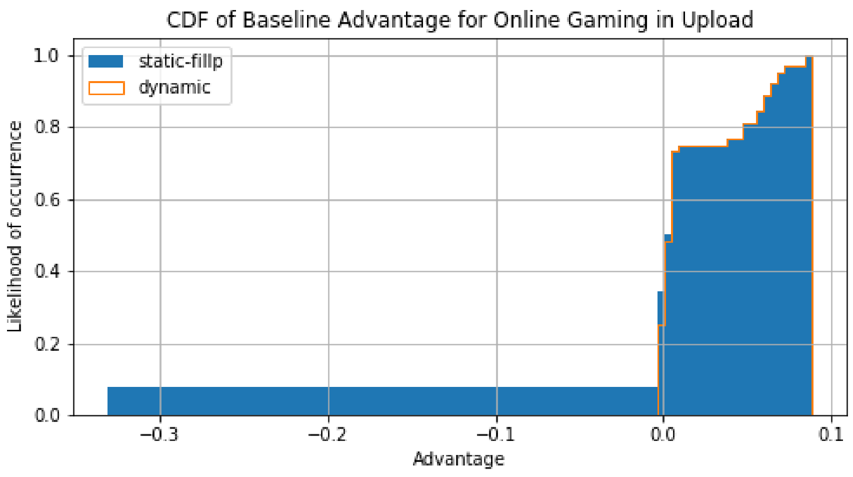
Figure A.16: CDF for application utility in download



(a)

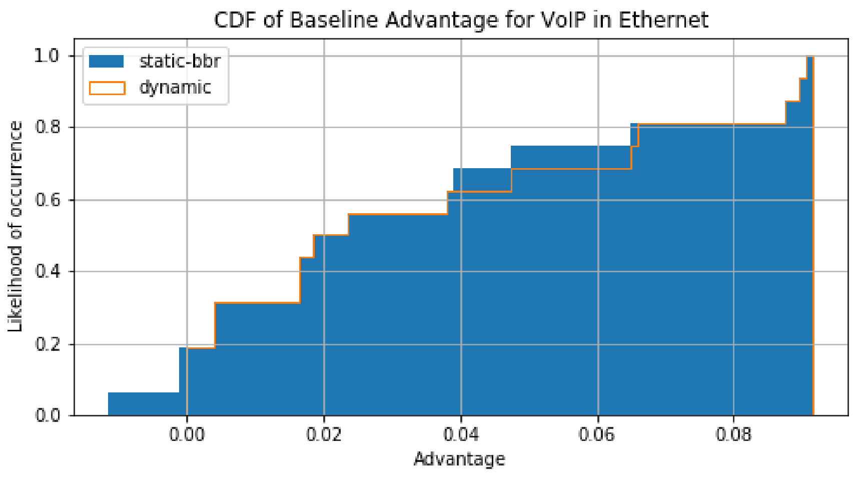


(b)

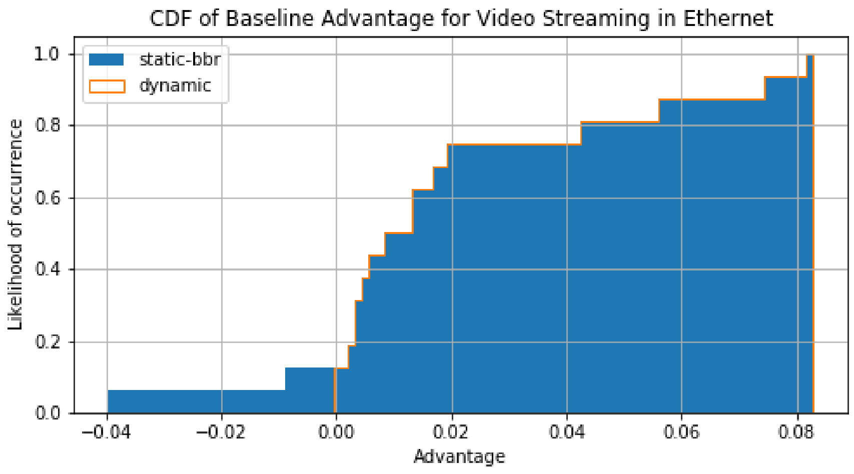


(c)

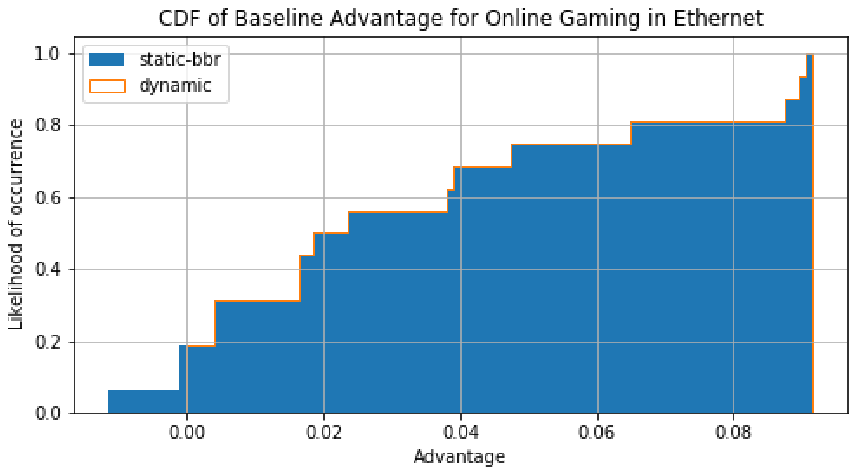
Figure A.17: CDF for application utility in upload



(a)

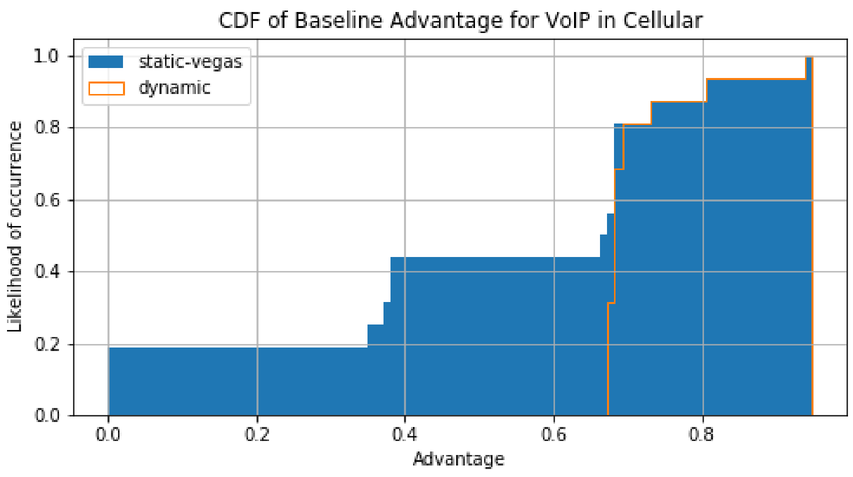


(b)

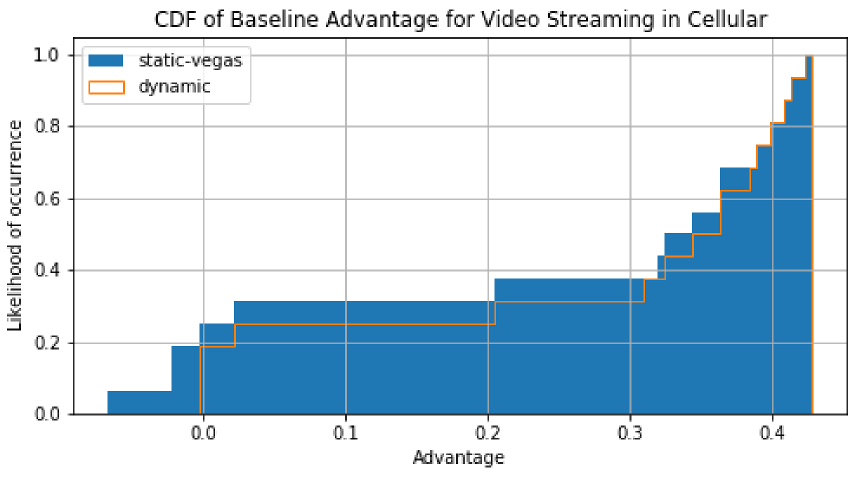


(c)

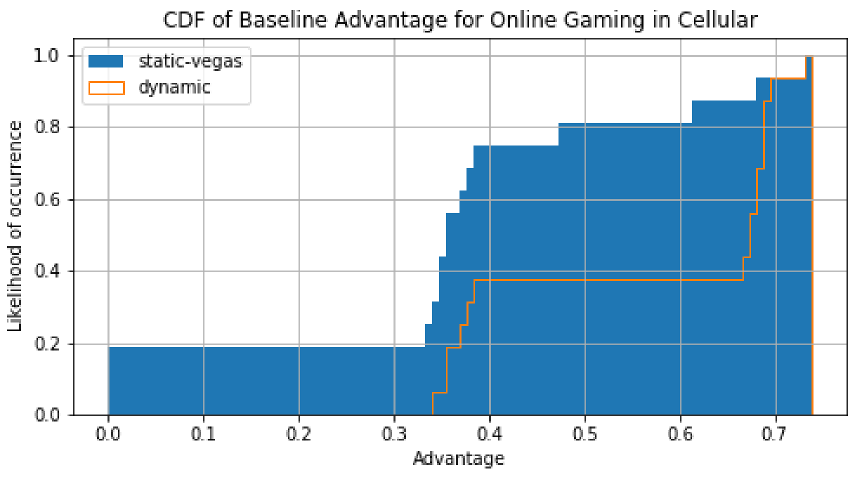
Figure A.18: CDF for application utility in ethernet



(a)



(b)



(c)

Figure A.19: CDF for application utility in cellular

A.2 Tables

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	93.75
Throughput-only	FillP	95.83
Loss-only	Indigo	33.33
Delay-and-throughput	FillP	85.42
VoIP	Vegas	33.33
Video Streaming	Vegas	33.33
Online Gaming	Vivace	29.17

Table A.1: Static Best algorithms for different metrics in London. The percentage column shows in which portion of measurements the algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	95.83
Throughput-only	FillP	100.0
Loss-only	Vegas	45.83
Delay-and-throughput	FillP	89.58
VoIP	Vegas	33.33
Video Streaming	FillP	43.75
Online Gaming	FillP	52.08

Table A.2: Static best algorithms for different metrics in Iowa. The percentage column shows in which portion of measurements the algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	95.83
Throughput-only	FillP	100.0
Loss-only	Vegas	35.42
Delay-and-throughput	FillP	97.92
VoIP	TCP Cubic	50.00
Video Streaming	TCP Cubic	39.58
Online Gaming	FillP	70.83

Table A.3: Static best algorithms for different metrics in Tokyo. The percentage column shows in which portion of measurements the algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	85.42
Throughput-only	FillP	77.08
Loss-only	QUIC Cubic	72.92
Delay-and-throughput	FillP	91.67
VoIP	Vivace	31.25
Video Streaming	TCP Cubic	39.58
Online Gaming	FillP	47.92

Table A.4: Static best algorithms for different metrics in Sydney. The percentage column shows in which portion of measurements the algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	Vivace	78.13
Throughput-only	TCP Cubic	90.63
Loss-only	Vivace	40.63
Delay-and-throughput	Vivace	62.50
VoIP	TCP Cubic	90.63
Video Streaming	Vegas	53.13
Online Gaming	TCP Cubic	90.63

Table A.5: Static best algorithms for different metrics in Brazil-AWS Brazil node pair. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	93.75
Throughput-only	FillP	100.0
Loss-only	QUIC Cubic	68.75
Delay-and-throughput	FillP	81.25
VoIP	Vegas	56.25
Video Streaming	FillP	50.00
Online Gaming	FillP	93.75

Table A.6: Static best algorithms for different metrics in Iowa-Tokyo node pair. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	100.0
Throughput-only	FillP	100.0
Loss-only	Vegas	50.00
Delay-and-throughput	FillP	93.75
VoIP	TCP Cubic	68.75
Video Streaming	FillP	43.75
Online Gaming	FillP	93.75

Table A.7: Static best algorithms for different metrics in Tokyo-Sydney node pair. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	75.00
Throughput-only	FillP	87.50
Loss-only	QUIC Cubic	75.00
Delay-and-throughput	FillP	93.75
VoIP	Vivace	50.00
Video Streaming	TCP Cubic	37.50
Online Gaming	Vivace	50.00

Table A.8: Static best algorithms for different metrics in Sydney-London node pair. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	28.13
Throughput-only	FillP	57.81
Loss-only	Vivace	42.19
Delay-and-throughput	Indigo	53.13
VoIP	TCP Cubic	40.63
Video Streaming	Vivace	35.94
Online Gaming	FillP	37.50

Table A.9: Static Best algorithms for different metrics in download. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	Vivace	42.19
Throughput-only	Indigo	42.19
Loss-only	Copa	18.75
Delay-and-throughput	FillP	29.69
VoIP	FillP	40.63
Video Streaming	FillP	40.63
Online Gaming	FillP	40.63

Table A.10: Static best algorithms for different metrics in upload. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static Best	Percentage (%)
Delay-only	QUIC Cubic	93.75
Throughput-only	FillP	100.0
Loss-only	QUIC Cubic	56.25
Delay-and-throughput	Indigo	43.75
VoIP	BBR	75.00
Video Streaming	BBR	87.50
Online Gaming	BBR	81.25

Table A.11: Static best algorithms for different metrics in ethernet. The percentage column shows in which portion of measurements the static best algorithm performs the best in all algorithms.

Metric	Static baseline	Percentage (%)
Delay-only	Indigo	50.00
Throughput-only	Verus	31.25
Loss-only	Vegas	43.75
Delay-and-throughput	Vegas	31.25
VoIP	Vegas	43.75
Video Streaming	Vegas	62.50
Online Gaming	Vegas	43.75

Table A.12: Static best algorithms for different metrics in cellular. The percentage column shows in which portion of measurements the static best performs the best in all algorithms.