Config-Snob: Tuning for the Best Configurations of Networking Protocol Stack

Manaf Bin-Yahya, Yifei Zhao, and Hossein Shafieirad, *Huawei Technologies Canada;* Anthony Ho, *Huawei Technologies Canada and University of Waterloo;* Shijun Yin and Fanzhao Wang, *Huawei Technologies China;* Geng Li, *Huawei Technologies Canada*

https://www.usenix.org/conference/atc24/presentation/bin-yahya

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Open access to the Proceedings of the 2024 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Config-Snob: Tuning for the Best Configurations of Networking Protocol Stack

Manaf Bin-Yahya[1], Yifei Zhao[1], Hossein Shafieirad[1], Anthony Ho[1,3], Shijun Yin[2], Fanzhao Wang[2], and Geng Li[1,*]

[1]Huawei Technologies Canada
[2]Huawei Technologies China
[3]University of Waterloo

Abstract

Web servers usually use predefined configurations, yet empirical studies have shown that performance can be significantly improved when the configurations of the networking protocol stack (e.g., TCP, QUIC, and congestion control parameters) are carefully tuned due to the fact that a "one-size-fits-all" strategy does not exist. However, dynamically tuning the protocol stack's configurations is challenging: first, the configuration space is ample, and parameters with complex dependencies must be tuned jointly; second, the network condition space is also large, so an adaptive solution is needed to handle clients' diversity and network dynamics; and finally, clients endure unsatisfactory performance degradation due to learning exploration. To this end, we propose Config-Snob, a protocol tuning solution that selects the best configurations based on historical data. Config-Snob exploits the configuration space by tuning several configuration knobs and provides a practical fine-grained client grouping while handling the network environment dynamics. Config-Snob uses a controlled exploration approach to minimize the performance degradation. Config-Snob utilizes causal inference (CI) algorithms to boost the tuning optimization. Config-Snob is implemented in a QUIC-based server and deployed in a large-scale production environment. Our extensive experiments show that the proposed solution improves the completion time over the default configurations by 15% to 36% (mean) and 62% to 70% (median) in the real deployment.

1 Introduction

User experience is one of the most valuable metrics that impact web services. Many studies show that a very small improvement in the Page Load Time (PLT) can lead to a significant impact on user experiences [22]. In the protocol stack, there are plenty of configuration knobs that can be tuned to improve the user experience. Web servers usually use predefined choices for the configuration knobs' values [17]. The

*Corresponding author: Geng Li (ligeng23@huawei.com)

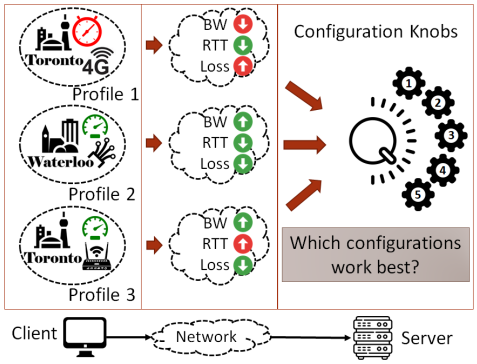


Figure 1: Problem Overview: the server receives numerous diverse client requests that endure different network conditions. The server has to set the best configurations for each request.

"One-configuration-fits-all" strategy does not suit all network scenarios. Initial configurations greatly impact not only the starting phase performance of the web services (e.g., connection establishment time) but also the overall performance (e.g., completion time). The initial configurations are often used in the connections (e.g., paces restart every time applications go idle) [10]. Different initial pacing rates and initial *cwnd*s can result in different performances depending on the network conditions. Manual protocol tuning [2] is a non-flexible and static approach that is constructed based on observations to meet limited requirements.

Learning the best configuration (i.e., protocol tuning) is a challenging task due to network dynamics, client diversity, and large configuration space. Fig. 1 shows an example of a client/server scenario that gives an overview of protocol tuning challenges. Diverse clients (based on, e.g., geolocation, ISP, network interface) face different network conditions (e.g., bandwidth, packet loss rate, and RTT), which affect the performance; thus, the effectiveness of protocol tuning. Each scenario has its optimal configurations. Tremendous efforts have been made to find the optimal configura-

tions [1–4, 15, 26, 29, 32, 33, 47, 48]. However, none of them fulfill all the requirements of generality, adaptability, stability, scalability, low overhead, and reliability. **Generality:** Configuration types and their knobs' ranges vary from one system to another [17]. The solution must support a wide range of configuration knobs with complex dependencies [29]. Some existing approaches are very specific to the knobs; thus, they cannot be applied to jointly tune a different set of knobs [4, 32, 33]. **Adaptability and Stability:** The solution must provide the best performance across fluctuating network conditions and sometimes performance metrics [8]. **Scalability:** The number of flows/clients connected to a server might be large. As a result, the solution cannot deal with connections individually, as it typically results in a few data samples for learning, along with higher processing overhead. **Overhead:** The search space is very large, and searching for the best configuration is expensive [3]. Also, extracting, processing, and returning the configuration in a timely manner is required. **Reliability:** Solutions usually endure performance degradation, and they cannot learn agilely during exploration due to the random or predefined trials [3, 29].

In this paper, we propose Config-Snob that tunes a set of protocol stack's configuration knobs to improve the performance (e.g., completion time). Config-Snob leverages the Multi-armed Bandit (MAB) approach; it trains Bayesian Optimization (BO) models that can be used to select the best configurations for the received connections. Config-Snob is a black-box solution that is not tied to certain configuration types or knobs. In order to handle the high-dimensional client features and variable network conditions, a two-level client profiling approach is used to help find a suitable model for a given client. To deal with new clients, initially, Config-Snob uses configurations that result from training a model using similar clients' data to less enduring performance degradation. Additionally, a controlled exploration approach is used to cope with the exploration cost by limiting the search space to a subset that has been evaluated to have a positive impact. To deal with unstable and fluctuating performance metrics that are used as rewards in the learning model due to a small number of samples, a causal inference algorithm is used [5].

Config-Snob is implemented in a QUIC-based application server (Nginx). The solution is deployed in a large-scale production environment, with 3M to 19M user requests per day. It is demonstrated that Config-Snob reduces the completion time over the default configurations by 15% to 36% (mean) and 62% to 70% (median). Additionally, Config-Snob is evaluated through an oracle testbed of client/server sessions over simulated network tunnels (i.e., FCC [16], cellular networks [28, 43], and Pantheon [44]). Config-Snob is compared with configanator [29] and cherrypick [3]. The results demonstrate that Config-Snob overcomes the challenges of data-driven protocol tuning. For the oracle testbed scenarios, Config-Snob improves the completion time by 35% to 49% (mean). The number of performance degradation cases is minimized to 0 out of 94K cases for the FCC and cellular network tunnels. At the same time, performance degradation is 87% less than other approaches for Pantheon traces.

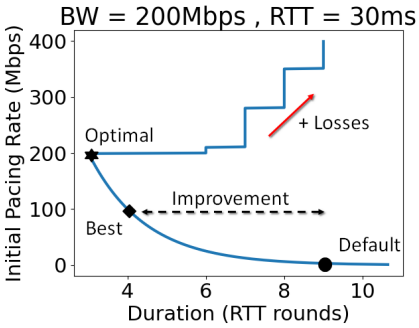


Figure 2: Theoretical analysis of the effect of Initial Pacing Rate on the startup (and drain phase) duration of a scenario of actual BW = 200Mbps and RTT = 30ms.

Our main contributions can be summarized as follows:

- By proposing Config-Snob, we demonstrate that connection completion time can be improved by using an adaptive "Black-box" solution to tune a set of configuration knobs. A controlled exploration is used to minimize the performance cost of the black-box approach.
- Config-Snob handles clients' diversity and network dynamics using a two-level fine-grained client profiling. Similar clients are classified based on their profile information and historical network statistics.
- To the best of our knowledge, Config-Snob is the first protocol tuning solution that utilizes causal inference (CI) algorithms to boost the optimization system. A causal forest (CF) algorithm is used to limit the search space and estimate the inaccurate reward.
- Config-Snob is deployed in a QUIC-based server. Config-Snob achieves comparable completion time improvement over the default, which is approximately near to the optimal configurations. Moreover, it achieves dependable performance in a large-scale production environment.

2 Background and Motivations

2.1 Problem Analysis

The one-size-fits-all strategy for setting the protocol stack configurations cannot adapt to the diversity of cloud networks. For example, geolocation has a large impact on web performance. Clients in developing regions endure worse network conditions than in developed regions [17]. For example, in China, the average delay of the first video frame is smaller

Table 1: Comparison between Config-Snob and related works

	TCP-RL [33]	**Cherrypick**** [3]	**Configanator** [29]	**Config-Snob**
C1	2 knobs (IW, CC)*	Set of knobs	Set of knobs	Set of knobs
C2	Multiple features	N/A	Multiple features + NC*	Two-level of client grouping
C3	RL agent with predefined initials	BO exploration with random sampling initials	BO exploration with random sampling initials	Controlled-BO exploration
C4	No	No	No	Yes

* CC = Congestion Control Protocol, IW = Initial Congestion Window, NC = Network Classification.
** Cherrypick is used to set cloud setup-related configurations that doesn't deal with clients' connections directly.

than 400*ms*, while in Southeast Asia, it is smaller than 650*ms*. On the other hand, the internet speed is continuously evolving, which leads to an unutilized room in the network capacity [41]. Also, bandwidth can be a huge difference between ISPs/network access interfaces [16]. Moreover, connections at peak time face more severe network conditions compared with off-peak time. In addition, the congestion control algorithm's performance can vary based on the device types (e.g., running BBR in mobile devices might lead to poor goodput [40]). Thus, each scenario can have its optimal configurations.

As an example, the best initial pacing rate and initial *cwnd* depend on the scenario's network conditions. Connections have the highest goodput and lowest RTT whenever the pacing rate equals bottleneck bandwidth, and the data in flight is equal to the bandwidth-delay product (full pipe) [9]. Operating at this optimal point maximizes the delivered bandwidth while minimizing delay and loss. Thus, the performance improved (i.e., fast convergence) if the connection starts with the pacing rate and initial cwnd that meet this optimal point. In BBR, the algorithm searches for the optimal point in the startup phase and then drains the resulting queue [9]. Still, most of the internet sessions are short flows which means the duration of startup and drain phases is influential to the flow completion time [7]. The startup duration for estimating the bottleneck bandwidth can be shortened using a near equivalent initial pacing rate (as shown in Fig. 2). Also, increasing the initial *cwnd* will reduce the number of RTT rounds to send the data in the startup phase. However, larger values than the optimal point lead to packet losses and higher RTT due to retransmissions because it exceeds the pipe capacity.

2.2 Challenges

The design of a data-driven protocol tuning solution encounters several challenges. Table 1 shows a comparison between Config-Snob and some of the most related work. The motivation of Config-Snob is to handle these challenges:

C1: Utilizing large configuration space. Most of the works in the literature do not utilize the configuration knobs and just tune one or two knobs [1,4,33,47,48]. Only the configanator utilizes a set of configuration knobs for TCP and HTTP protocols [29]. The protocol stack has a large and diverse configuration stack. As a result, we propose a "Black-box" solution that jointly tunes a set of configuration knobs with complex dependencies. Searching for the best configuration is expensive due to the large configuration space. Thus, we evaluate the impact of configurations on different profiles using the causal relationships between the network conditions.

C2: Handling diverse clients. Dealing with connections individually usually leads to few data samples [1]. Having fine-grained client grouping using only IP-prefix will not overcome the shortage problem [33]. To address client space diversity, we propose a two-level client grouping approach. This ensures that each group has sufficient samples and effectively handles the dynamic network environment by classifying similar profiles.

C3: Handling exploration cost. Randomly selecting configurations results in poor performance and typically suits a small and limited decision space [26,33]. Existing approaches highly suffer from performance degradation in the exploration phase [3,29]. Clients cannot afford such poor performance even for a few connections before getting the optimal or near-optimal configurations. Thus, the number of negative experiences is significant when applying the tuning solutions. We exploit the historical configuration performance to bootstrap the exploration with the best suitable configurations from overlapping profiles. Furthermore, we use a function that assists the exploration by excluding a subset of the space that might have a negative impact on the outcomes (i.e., performance degradation).

C4: Dealing with inaccurate rewards. The average reward is calculated to evaluate the impact of configurations, which may not be accurate due to randomness and complex dependencies between data samples, as well as insufficient data in some realistic scenarios. For example, short flows are usually ended before collecting precise network statistics or applying any control decisions [14,33,48]. As a result, the learning models are trained with noisy and loose data, and it becomes difficult to train (predict) the best configurations. We use a re-

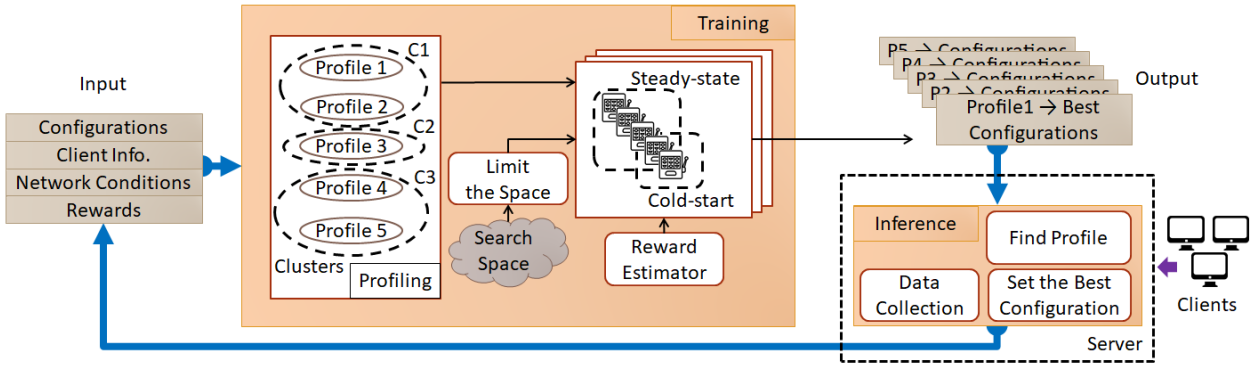


Figure 3: Config-Snob Design Overview.

ward estimator technique that provides an efficient and robust prediction of the impacts of the changes in the configurations by using the causal relationships among the network conditions.

2.3 Config-Snob

Below, we describe some of the design choices to handle the aforementioned challenges:

Profiling (§3.1). The main purpose of profiling is to achieve scalability as it's impractical to fine-tune configurations individually for each connection. Our profiling approach tackles additional challenges encountered in large-scale deployments, including dynamic network conditions and efficient learning. Initially, we use fixed features to group requests into profiles. Profile features such as geolocation (city), ISP, IP subnet, and connection time are used to provide context continuity. However, profiles may result in a smaller number of samples, which is suboptimal for training. To address this, we employ a clustering method to create profile groups characterized by similar network conditions, allowing for joint training.

Bayesian Optimization (§3.2.1). Config-Snob uses BO as the learning model because it provides black-box optimization that can adaptively explore the search space and find the best set of parameters that maximize the objective function with limited data using a prior probability function.

Multi-Armed Bandit (§3.2.2). Config-Snob uses MAB to handle the exploration-exploitation trade-off. One arm is used to perform controlled exploration when the profile group does not have enough samples or the reward distribution is not stable. Multiple arms are used during the steady state to exploit the best predictions and continue the exploration to avoid local optimum but in a conservative manner.

Causal Inference (§3.3). Config-Snob uses CI to precisely estimate rewards and to ensure that the selected configurations for new connections are less prone to negative experiences. The average reward may not be accurate due to correlated noise, randomness and complex dependencies between data samples, and insufficient observational data, especially in real-time scenarios. The CI-based reward estimator, by predicting the impacts of the changes in the configurations, improves the accuracy of the calculated reward. Moreover, CI is utilized to refine the search space by excluding configurations that detrimentally impact performance.

3 Config-Snob Design

The central principle of Config-Snob is to utilize historical data in order to select the best configurations for connections considering diverse client features and network conditions. The decision is made per profile group; thus, Config-Snob holds several models. Fig. 3 shows the design overview.

3.1 Profiling

To address the challenge of the diverse network scenarios, Config-Snob applies the learning models at the profile group granularity. First, different profile features (e.g., geolocation, ISP, network interface, connection speed, OS, IP subnet, and connection time category) are combined to create a profile as they are most likely to face very similar network conditions. Second, profiles that have similar network features are grouped in clusters to produce profile groups that are used for the learning model. We assume that the network conditions of one profile are stable within some time (some updating cycles). The main idea is to apply online exploration-exploitation for each profile group (§3.2).

3.1.1 Profile Tree

A tree-based design is used for the profiling that is easy to train and create overlapping groups, and has efficient inference. The profile tree is created and updated (constantly) based on collected samples from the previous updating cycle. Each level represents a predefined profile feature. Thus, a path to a leaf node represents a profile (sample IDs are stored in the leaves). Algorithm 1 gives an overview of level 1 profiling

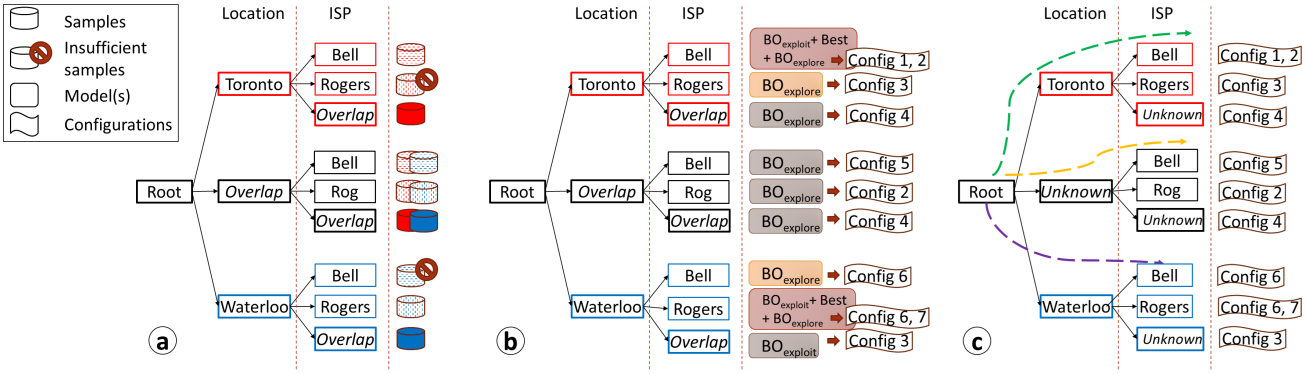


Figure 4: Example of profile tree; (a) Creating profile tree including *overlap* nodes and determining sufficiency; (b) Learning models based on the profile type and its sufficiency. (c) Traversing the tree to get the assigned configurations (inference phase).

steps. In Fig. 4, the profile features are location and ISP (for simplicity). A path to a leaf node represents a profile. At the training, profiles are associated with learning models, while profiles are associated with predicted configurations at the inference. In Fig. 4(c), the green line represents a connection with a profile (Toronto, Bell) which is already seen in the trained data; thus, Config 1 and Config 2 are returned after traversing the tree.

Algorithm 1 Level 1 Profiling Algorithm

```
if Profile Tree not exist then
    create Profile Tree;
end if
for each profile ∈ Profiles do
    if Profile exist then
        insert profile samples;
        update overlap groups;
    else
        create profile;
        insert profile samples;
        create/update overlap groups;
    end if
end for
```

3.1.2 Unseen Profiles

Some connections might come from unseen clients (i.e., there is no profile yet for these clients). In Config-Snob, the initial configurations for these clients are the outcome of BO models that are specifically trained by samples with overlapping profile features (thus, closer network conditions). These overlapping groups are created during the training phase, and their outcomes are used for new profiles' inference. Note that these extra groups are not the most fine-grained; thus, these groups do not operate in the exploitation mode. The *overlap* nodes are labeled as "Unknown" nodes in the inference. In Fig. 4(a), (Toronto, *overlap*) is a group that combines (Toronto, Bell) and (Toronto, Rogers) samples. While (*overlap*, *overlap*) is a combination of all seen profiles. In Fig. 4(c), the yellow line represents a connection with unseen features such as (Ottawa, Bell), thus, Config 5 will be used.

3.1.3 Sufficiency

Config-Snob checks the profile's sufficiency to ensure that adequate historical data is collected to provide steady decisions. A profile is sufficient when there are enough samples (at least k samples) to train and the reward distribution is stable over time. A reward constraint from TCP-RL [33] is used. On the other hand, an insufficient profile shapes a profile group alone. In Fig. 4(b), (Waterloo, Bell) profile is insufficient; thus, it will continue/move to the exploration phase.

3.1.4 Clustering

Clustering profiles with similar network conditions can improve performance. Config-Snob uses network features (i.e., goodput, packet loss rate, and RTT) to cluster the profiles using the k-means algorithm. The output of clustering is assigned to profile groups. Profile groups are groups resulting from clustering and overlapping groups.

3.1.5 Time Category

We split the day and time into several time categories. In the current design, the time categories are *weekday offPeak*, *weekday Peak*, and *weekend* (anytime on the weekends and holidays). We observe that during the same time category, the network conditions and the number of received connections are similar. However, sharp changes (spark) in the network might occur during the *Peak* or *offPeak* time categories (e.g., peaks may occur in the *offPeak* time category and valleys in the *Peak* time category). The spark duration is usually short (in time) but can last for a few updating cycles. As a result, when the spark is detected during the training process, the solution uses the previous time category models' outcomes

for the inference. For example, if a valley is detected during the *Peak* time category, the solution will not train the *Peak*-specific (profile) models with the new samples; otherwise, it will use the stored *offPeak* models' outcomes and send it for inference in the next updating cycle.

3.2 Learning Model

Learning models are created for each profile group and trained to generate the predicted configurations. There are two phases in the training cycle. At the exploration (cold start) phase, the data is collected to learn about the network distribution for a certain profile and explore the search space. At the exploitation (steady-state) phase, the profile must satisfy the sufficiency check, and the solution fully utilizes data from previous updating cycles to find the optimal configurations.

3.2.1 Bayesian Optimization

Our problem formulation is to find the optimal values for a set of configuration knobs, denoted as θ. The objective is to maximize the reward function (based on completion time), which is denoted as $f(\theta)$. f is the mapping function that maps configuration θ from the search space Θ to the reward. Bayesian optimization (BO) is a strategy for the global optimization of black-box functions [37]. We use BO in the granularity of the profile group to consider multiple parameters (profile features and network conditions). BO can adaptively explore the search space and find the optimal points with limited data. The BO model starts with an initial set of values, which can be random (e.g., LHS) or based on prior knowledge (e.g., *overlap* group model). Furthermore, a set of box constraints (i.e., the CF limited search space) is defined as $\hat{\Theta} = g(\Theta, \theta)$ where $\hat{\Theta}$ is the limited space, and g is the CF function that limits the search space (§ 3.3.3).

3.2.2 Multi-Armed Bandit

Config-Snob uses a four-armed bandit. **Case 1:** The profile is sufficient (steady-state); three arms are used to generate configurations (exploitation arm (Best), exploitation arm (BO Model 2), and exploration arm (BO Model 3)). The two BO models are utilized to mitigate the overfitting issue that might be introduced by profiles. In the inference, different frequencies are assigned to each arm outcome (0.5, 0.4, and 0.1, respectively). **Case 2:** The profile is insufficient (cold start); an exploration arm (BO Model 1) is used to generate next cycle configurations. **Case 3:** for overlapping groups that are used for new profiles, an exploration arm (BO Model 1) is used to generate initial trials for new profiles. Prior knowledge (i.e., *overlap* group model) is used as initial trials for the new profiles. The goal is to minimize performance regret and focus the search on promising regions of the space. This set of initial trials is created from similar profiles, and it is regularly updated and explored.

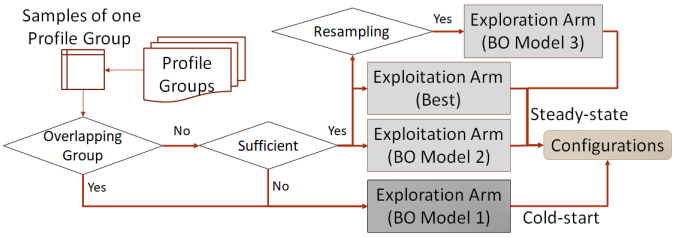


Figure 5: Config-Snob Learning Models (MAB).

Fig. 5 shows the MAB learning flowchart for each profile group. In Fig. 4(b), (Toronto, Bell) and (Waterloo, Rogers) profiles are sufficient groups, thus, the steady-state models are used. However, (Toronto, Rogers) and (Waterloo, Bell) profiles are insufficient; thus, only the exploration arm is used. Finally, overlapping groups use exploration BO models.

Exploration Arm (BO Model 1). This BO model is used for the exploration process. First, each overlapping group will train a specific model. The outcomes will be used for the new profiles to create a set of initials (based on prior knowledge) for their specific BO model. Second, for seen profiles, if the profile group already has enough samples, then use it to train its BO model to do further exploration; otherwise, use the previous cycle inference.

Exploitation Arm (BO Model 2). This exploitation arm is the first steady-state arm in the MAB. This BO model uses a smaller step size than the one in the exploration arm.

Exploitation Arm (Best). We use the best-seen configurations so far (based on the reward) for the profile group in the inference. We re-evaluated it every few updating cycles.

Exploration Arm (BO Model 3). Apart from the two exploitation arms, in order to tackle non-Gaussian noises and further utilize the recent history records, BO Model 3 with a resampling mechanism is used to do moderate exploration in the steady phase under specific conditions [29]. Resampling in a steady state happens for each profile group if the number of configurations tried (seen) by the BO exploitation arm is greater than a threshold. Thus, another BO model is created for exploration with larger step sizes. This BO model is initialized with the top five configurations (largest rewards).

3.2.3 Reward

We argue that the reward for such a solution must be a performance metric that is closely tied to the user experience, such as PLT in the web services and Time to First Frame (TTFF) in video services. In Config-Snob design, we use the connection completion time (CT). During the training phase, the reward is calculated for each profile group. This calculation is based on all requests' data that belong to the group and were gathered from the previous cycle. Thus, the reward is as follows:

$$R_t^G = \frac{\sum_{f \in F_G} \frac{CT_f}{size_f}}{len(F_G)} \quad (1)$$

where G is the profile group, t is the updating cycle, F_G is the connection set of group G, CT_f and $size_f$ are completion time and sent data size of connection f, respectively. In some cases, averaging the reward is not efficient due to the fluctuation within the same group. Thus, we propose a reward estimator method based on the CI method (§ 3.3.4). Furthermore, to determine the profile sufficiency, the reward distribution factor for a profile is calculated to measure the changes in the normalized reward between two consecutive updating cycles [33].

3.3 Causal Inference

Causal inference (CI) is a statistical technique that allows us to ascertain the causal relationship between different problem variables, as well as identify the impact of a specific variable (called treatment) on an outcome, while accounting for the influence of other variables, referred to as confounders and covariates [45]. Although the optimization tools can provide valuable insights into the optimal configuration of protocol parameters, they may not accurately evaluate the impact of configuration changes on performance. In this regard, CI techniques can be used to estimate the causal effect of parameter changes on the outcome of interest, while accounting for the impact of other potential variables. The CI techniques can also enhance our understanding of how modifications of different parameters affect the reward. As an extension of the random forest algorithm, Causal Forest (CF) algorithm [5] is designed explicitly for CI learning tasks, as an efficient tool, used in this work.

3.3.1 Methodology

The CF's procedure comprises six key steps: 1) Initiate the process by specifying a treatment, which consists of a sequence of N configuration parameters for assessing their impact on the reward, 2) Prepare the dataset through data normalization and standardization techniques, 3) Efficiently select features from the network logs dataset (referred to as covariates) using the combination of the CovSel [18] algorithm and domain expertise, 4) Assess the overlap between the treatment and control groups as a necessary requirement for CI analysis, and apply matching techniques to the dataset before utilizing the CF model, 5) Train a tuned CF algorithm using the pre-processed and matched data to estimate the Conditional Average Treatment Effect (CATE), indicating expected difference in potential outcomes under various treatment conditions, considering observed covariates, and 6) Evaluate the outcomes based on the ground-truth (GT), which here, represents the average reward obtained from the dataset .

To implement these steps, data is initially collected for one day, and all six steps are executed at the end of the first day. Starting from the second day, all steps are repeated daily, except for the feature selection part (step 3), which is performed weekly to accommodate potential changes in feature importance. Concerning steps $3-4$, in CI, obtaining unbiased treatment effect estimates necessitates comparable treatment and control groups. Ensuring overlap, where each covariate level includes both treated and control units, is crucial [19]. Without overlap, causal estimates may be skewed, reflecting covariate differences rather than true treatment effects. Covsel, a covariate selection method, plays a vital role by identifying covariates that significantly influence treatment assignment and outcomes.

Following Covsel, matching is performed to pair similar treated and control units, further enhancing comparability and reducing bias. For feature selection, the initial step involves using the Covsel tool, followed by filtering the selected covariates based on domain expert knowledge. This process selects covariates such as maximum estimated bandwidth, minRTT, and bytes sent. Matching is achieved through the propensity score matching techniques, which attempts to estimate the treatment effect while accounting for covariates predicting treatment reception [25]. Hence, ensuring overlap, covariates selection, and matching are pivotal components of CI.

3.3.2 Causal Forest Algorithm

CF algorithm, renowned for its versatility in handling various types of features, is an efficient tool for CI learning. It achieves this by constructing a forest of decision trees, a technique that ensures unbiased and robust results. To enhance accuracy and mitigate overfitting, CF employs random feature subsets during the tree-building process. The splitting criteria in CF focuses on outcome variations between treatment and control groups, considering other relevant factors to optimize the identification of group differences. Our goal here is to estimate the Heterogeneous Treatment Effect (HTE). HTE unveils the variance in treatment effects across different subsets of the population, enabling us to discern how various scenarios are influenced by configuration parameters (treatments).

In this work, we opt for the CF algorithm as the cornerstone of our CI learning tasks, owing to its myriad advantages. Notably, the CF algorithm has consistently demonstrated a high level of accuracy across a wide array of applications. It is worth highlighting that the algorithm does not make any assumptions about the data distribution, rendering it a robust and adaptable choice for diverse datasets [5].

Given our emphasis on addressing realistic large-scale scenarios, scalability becomes a paramount advantage of the CF algorithm. Additionally, another key strength lies in the fact that the CF algorithm does not necessitate the variables under consideration [5]. This attribute proves especially valuable in complex networking scenarios where the precise relationship

between variables remains unknown. However, it is crucial to acknowledge that the CF approach, like many CI methodologies, operates under certain assumptions. These assumptions encompass concepts such as unconfoundedness or ignorability (ensuring that every sample has a positive probability of receiving either treatment, and given the covariates, the potential outcomes are independent of the treatment assignment). It is essential to recognize that these assumptions are common elements within most CI approaches.

3.3.3 Limited Search Space

Given the extensive configuration space Θ, we employ the *Limit the Space* module, utilizing CI techniques, to pinpoint and eliminate configuration values that typically result in undesirable outcomes, such as increased PLT or TTFF. This module operates by leveraging the trained CF and evaluating the CATE for each profile.

Using the current state of the configuration parameters as input, the module assesses the consequences (impacts) of transitioning to all potential future configuration knobs (future states). It identifies the future states that negatively impact the reward and subsequently pinpoints subsets of the configuration space likely to have an adverse effect on each profile. These identified subsets constitute the output and are removed, leaving the remaining areas as the permissible search space $\hat{\Theta}$ for the BO models.

3.3.4 Reward Estimator

The CI *Reward Estimator* module functions to generate reliable and precise estimates of rewards, specifically in situations where data samples are limited, thereby making the average reward inaccurate. Utilizing the trained CF algorithm, this module is employed to provide reward estimates for profiles with insufficient data. This method deviates from the conventional use of average rewards, substituting them with the CF-derived estimates, if the number of captured samples for each configuration, denoted by k, is less than 40. This module intakes specific profile samples and outputs the CATE corresponding to that profile, thereby streamlining the reward estimation process.

4 Implementation

We implement the Config-Snob in a QUIC-based application server that provides both web and video-based services to clients. The solution tunes a set of configuration knobs for QUIC sessions to improve the completion time performance (thus, PLT or TTFF). A modified version of the OpenResty server (Nginx platform) is used [35] to allow Config-Snob to assign configurations to the QUIC sessions. Fig. 6 gives an overview of Config-Snob's system architecture and workflow. The Config-Snob is written in Python and C++ with about 2500 lines of code, excluding the legacy part of the OpenResty server. The architecture of Config-Snob meets three deployment requirements (§5.3): **Scalability**: We tested it with over 800K connections per minute. **Responsiveness**: It responds to requests for inference within an average of 42 microseconds. **Fault tolerance**: The performance does not significantly impact when any system component fails.

4.1 APP Server

Config-Snob has five main modules in the APP Server that are developed in C++.

Client Information Parser. To enable Config-Snob's profiling, it must collect client information before establishing the QUIC session. Since the inference part of the solution is embedded into the APP server, we can extract client information (profile features) at the handshaking. The parser gets the IP address and obtains geolocation, ISP, and IP subnet by looking up existing IP databases [12, 27]. Furthermore, the interface type and the client OS are extracted from the SSL/TLS [30]. The time category is determined by the connection receiving time (§ 3.1.5).

Inferencing. The inferencing module takes the profile features from the parser as input to search the profile tree and return the configurations that will be used for the QUIC session. We use a pointer-based tree (a modified version of BTree) for profile tree implementation. It is faster in search (traversal); the time complexity is $O(l)$, where l is the number of profile features. Thus, our method is scalable over a large number of diverse clients. We set default configurations to initial random configurations generated by Latin Hypercube Sampling (LHS) [38]. It is only used if the prediction data does not exist or the interface fails to load the configurations.

Session Interface. We implemented an interface that allows Config-Snob to simply configure the server stack. This interface gets the selected configurations and sets them for the current session by modifying the callback function of setting up the QUIC session.

Data Collection. Config-Snob collects all the report data from the Nginx server and pushes it to the AI training server. Profile information, network statistics, used configurations, and performance metrics (completion time) are stored in the local database after the connection is closed. We use HTTP cURL to push the report data for the previous updating cycle.

Preprocessing. The inferencing module must have access to the updated profile tree associated with configurations that are received through the prediction receiver (pushed by the AI Training server using HTTP cURL) every updating cycle (around 6 minutes). An open socket is used to dispatch the prediction data from the AI Training server to the APP server. The inference profile tree is created if it does not exist; otherwise, update the current one based on the received data.

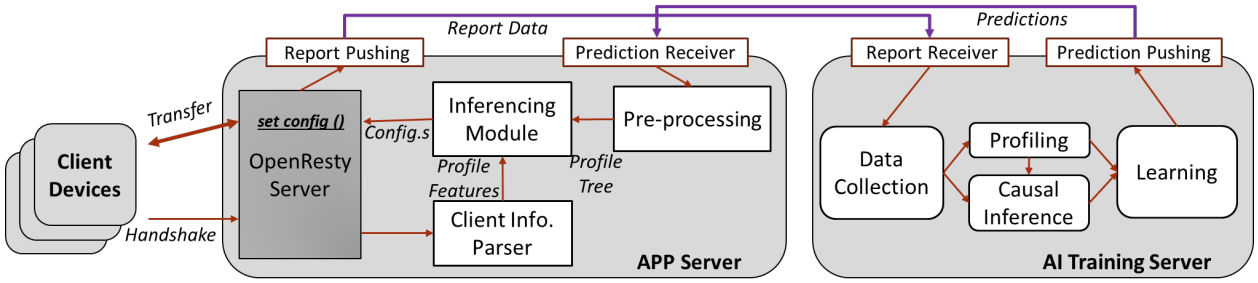


Figure 6: Implementation Overview of Config-Snob.

4.2 AI Training Server

Config-Snob has four main modules in the AI Training Server that are developed in Python. We use a workflow platform that orchestrates and schedules the task workflow and the data pipeline. Each module represents a task unit to be executed, including its dependencies.

Data Collection. We integrated the workflow platform with a compatible object storage library. It handles all the data pipelines in the architecture, including storing the report dataset. The report data coming from the APP server through the data receiver process are stored in this report dataset.

Profiling. Similar to the inference profile tree, Config-Snob uses modified B-tree implementation in the training phase. For clustering, Lloyd's KMeans algorithm [42] is used with a maximum number of iterations for a single run of 500 and a random state of 42 [36]. Config-Snob uses the scikit-learn machine learning library for k-means.

Learning Models. Open-sourced Bayesian Optimization package is used for BO implementation [34]. The initial number of samples in each user group for training is 20. The minimum number of configurations to switch from cold to steady state is 10. The minimum number of samples to enable BO resampling is 15. *xi* is set (0.1, 0.0001, and 0.007) for BO Model 1, 2, and 3, respectively.

Causal Inference. We use the CF algorithm from the EconML library [23] on the normalized, standardized and matched dataset, using the set of the most efficient covariates selected using the CovSel algorithm as well as domain expert knowledge. For matching, we use the propensity score matching technique. We run the CF algorithm using 100 trees, and the trained model is used to estimate the impact of configurations on the performance. For the training of the CF algorithm, we use the captured network logs, including the GT reward.

4.3 QUIC and Configuraion Knobs

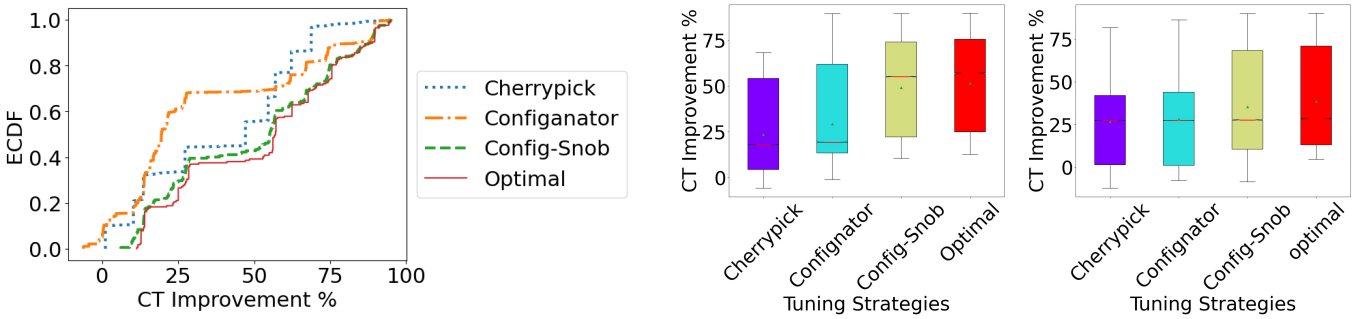
QUIC, as a cross-layer protocol, has plenty of configuration knobs that can be tuned to improve the user experience. We focus on tuning a set of 8 configuration knobs: (1-3) The first three knobs are initial pacing mechanism parameters (i.e., initial pacing rate and bytes, and if pacing stops when the first data is acknowledged). These parameters affect the duration of BBR phases, particularly the startup phase, which in turn impacts connection completion time. In §2.1, we discussed the influence of pacing mechanism parameters; (4) The buffered data threshold knob which specifies the size of the internal transmitting buffer on the server side for each single QUIC session; (5) Ping in tail configuration which is used when Tail Loss Probe (TLP) [20] algorithm is enabled; (6) Retransmission timeout; (7-8) The last two knobs are to enable the probe up mode of BBR.

5 Evaluation

To evaluate Config-Snob, we conduct large-scale experiments in both emulated and real-world environments. First, we use a large-scale trace-driven testbed to build an oracle. All experiments are QUIC sessions over simulated network tunnels (Mahimahi [31]). Second, we evaluate the real-world deployment of Config-Snob in a video-based service. Third, we evaluate the deployed solution's reliability, and last, we evaluate the design choices of our solution. In our experiments, we calculate improvement in CT relative to the CT of default configurations. Performance degradation means that the selected configurations result in a lower CT compared to the default configurations..

5.1 Oracle Testbed

Setup. The testbed is built based on FCC report [16] and cellular networks [28, 43]. Additionally, real-world trace-driven network tunnels are simulated (Pantheon [44]). QUIC clients send requests to the QUIC server via Mahimahi simulator tunnels which are configured with different sets of network conditions (i.e., latency, stochastic packet loss rate, and bandwidth traces). Clients connect to the server and load webpage files of different sizes (i.e., 50KB, 100KB, 200KB, 500KB, 1MB, and 2MB). We run five of client/server QUIC sessions for all of the configuration templates including the default



(a) FCC+ (ECDF). (b) FCC+ (box whiskers). (c) Pantheon (box whiskers).

Figure 7: Overall completion time (CT) improvement (oracle).

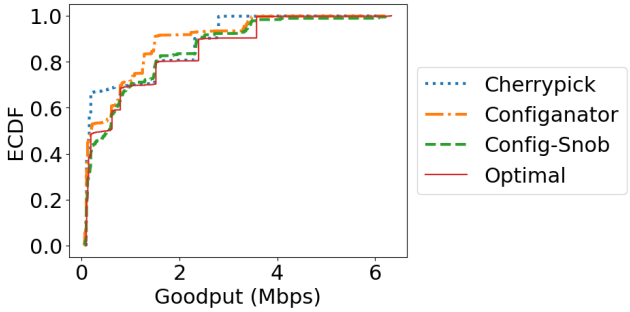


Figure 8: Distribution of goodput (oracle).

configurations, simulated network tunnels and webpage sizes (more details in Appendix A).

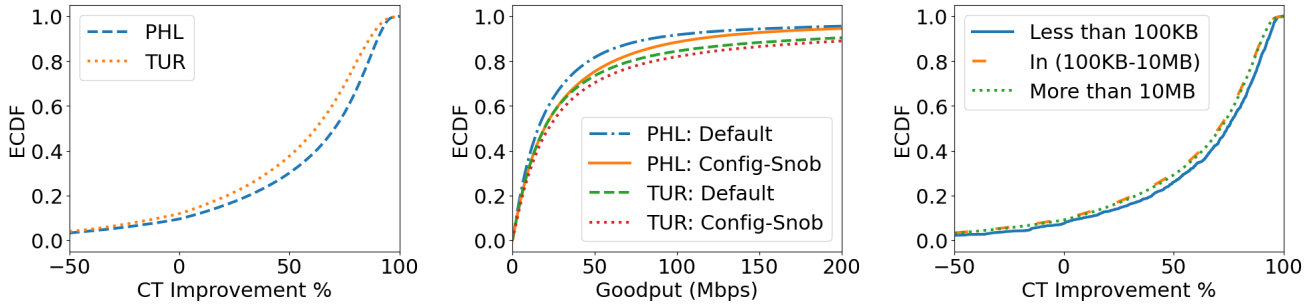
We compare Config-Snob performance with the optimal configurations (oracle) as well as different customized strategies. Cherrypick [3]: we implemented the strategy that selects the best configurations by exploiting the existing data for the seen profiles while random samples is used for the new profiles. GPyOpt [6] is used for BO implementation with initial samples = 3, expected improvement (EI) = 10%, and the minimum of samples tested = 6. Configanator [29]: we implemented the strategy that uses a decision tree to predict the configurations for existing profiles while it uses BO for exploration and LHS generates the initial trials. The Optimal is the set of configurations with the lowest completion time by brute-forcing the oracle.

Results. Fig. 7 compares Config-Snob with other strategies. Config-Snob improves the CT by 49% (mean) and 55% (median) over the default configurations for FCC+, and 35% (mean) and 28% (median) for Pantheon. The $95th$ percentile CT improvement reaches more than 90% in both setups. Cherrypick results into poor performance due to random exploration. Moreover, the two strategies use a limited profiling approach. The decision tree in the configanator improves the results when used in place of BO for the existing profiles. Config-Snob shows significant improvement due to the MAB model, two-level profiling approach, and the controlled exploration that starts with prior knowledge initials. All strategies endure some performance degradation due to the new profiles (§5.4). Besides, Fig. 8 shows that Config-Snob also outperforms other strategies in terms of the goodput, calculated by dividing CT by the total bytes sent.

5.2 Real-world Deployment

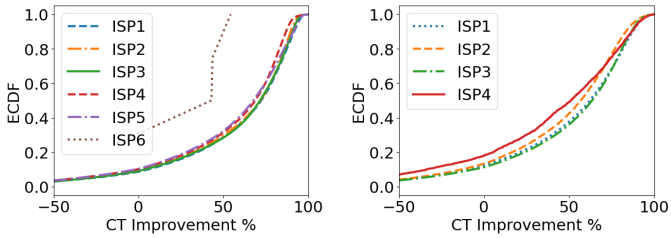
Setup. We deploy Config-Snob in a large-scale production environment of video-service application. The solution is deployed in two regions, namely Turkey (TUR) and Philippines (PHL). In TUR, it serves three edge nodes with $140K$ user requests per hour ($3M$ per day) and one edge node in PHL with $800K$ user requests per hour ($19M$ per day). Config-Snob has been deployed in this service for more than a month. The clients are diverse and endure dynamic network conditions. We configure the servers to keep using the default configurations for 20% of user requests. The default configurations are the predefined values in each server that were set previously by engineers.

Results. Fig. 9 shows the CT improvement of Config-Snob deployment. Config-Snob improves the CT for video sessions over the default configurations by 36% (mean) and 70% (median), and 15% (mean) and 62% (median) for PHL and TUR servers, respectively (Fig. 9a). TUR servers experience worse network conditions than the PHL server. Consequently, the solution converges to better predictions, leading to better performance for PHL. For the $10th$ percentile CT improvement, Config-Snob has 1.84% for PHL, while -8.94% for TUR. In general, we analyze the performance degradation cases which occur mostly due to harsh network conditions of some connections with high loss rates. Fig. 9b shows that Config-Snob achieves higher goodput in both regions. Fig. 9c represents the completion time improvement with different transfer size categories in both regions. Config-Snob reduces the overall completion time for small data transfers (less than $100KB$) by a median of 67% to 70% and 88% to 90% in the $90th$ percentile.



(a) Overall CT improvement. (b) Distribution of goodput. (c) Different transfer sizes.

Figure 9: Real-world deployment results.



(a) PHL. (b) TUR.

Figure 10: CT improvement for PHL and TUR ISPs.

Fig. 10 shows the performance of Config-Snob for a number of ISPs in both regions. Despite ISPs having different capabilities, Config-Snob reduces the completion time by a median of 43% to 72% and 51% to 91% in the 90th percentile for the PHL region. Although the PHY server receives very few connections from ISP6, Config-Snob shows a reasonable CT improvement. For the TUR region, it reduces the completion time by a median of 51% to 63% and 83% to 87% in the 90th percentile. TUR's ISP4 has very fluctuating network conditions which is reflected in the solution performance.

5.3 System Benchmark

We evaluate the scalability and responsiveness of our implementation. The experiments are performed in a testbed with an upper limit of 800K connections per minute involving 500 machines. The latency of the inferencing module is 0.042ms on average. The corresponding training time for the testbed is 3000ms on average. The latency of pushing the predictions through the interface is 300ms on average and a maximum of 2000ms. We tested reliability of the system against its components failure (e.g., the inferencing module will use the existing predictions when the AI Training server is not available).

5.4 Solution Choices

Setup. The same oracle testbed in §5.1 is used.

Profiling. Fig. 11 shows the goodput and average RTT compared to the solution without profiling (i.e., all connections are grouped and one BO model is used). Thus, the same configurations is used by the profiles every updating cycle). The profiling helps the models to learn efficiently, which improves the overall performance.

Controlled Exploration. Fig. 12 shows the results from the exploration phase of different strategies. Random approach shows a poor CT performance. Using the exploration approach in configanator shows a high number of performance degradation cases because the algorithm starts the exploration with a set of random trials. The utilization of existing data with overlapping profiles and the reduced search space provide Config-Snob stable results with lesser performance degradation cases. We achieved zero performance degradation cases out of 94K cases for FCC+. For Pantheon real-world traces, performance degradation is minimized by 87% compared to strategies that use random trials.

Reducing the Search Space. Table 2 demonstrates the impact of changing the configuration parameters from their default values to the new values, for different profiles with different BW values ranging from 25 to 100 Mbps. The results are averaged over each profile. To derive the CF estimated reward, denoted by R_{CF}, we employ the CF algorithm, utilizing a dataset of up to $k = 100$ gathered from each configuration. To evaluate the accuracy of the CF estimates, we compare R_{CF} with the GT reward, denoted by R_{GT}, which represents the average CT. We use Root Mean Square Error (RMSE) as the performance metric. For instance, an RMSE value of 0.01 for profiles 3 and 4 signifies an error of 1% in the CF estimated reward when compared to the GT reward. This outcome underscores the CF approach's capability to provide highly accurate estimates of configuration parameters impacts, which can be used for reducing the search space.

In this experiment, for each profile, we rank both the R_{GT} and R_{CF} for different configurations. We select the configurations which are negatively impacted (indicated by having the lowest rank), denoted by Min r_{GT} and Min r_{CF}, to access the CF's capability to accurately identify the configurations with

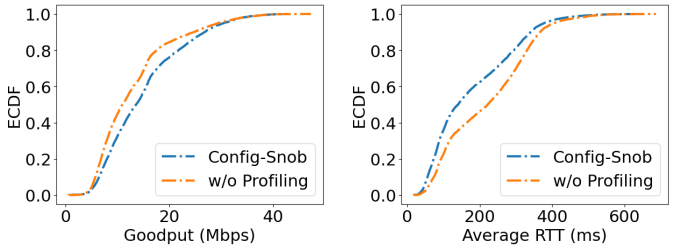
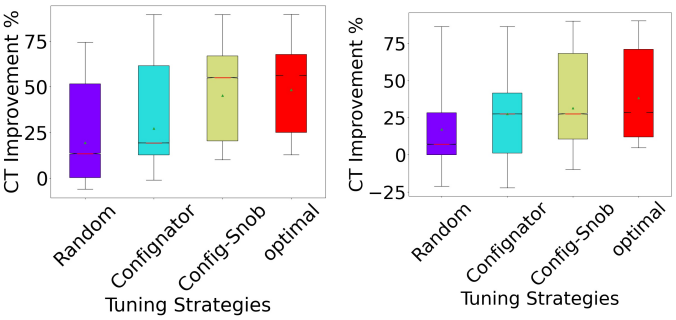


Figure 11: Goodput and average RTT with and without profiling.



(a) FCC+.

(b) Pantheon.

Figure 12: CT improvement in the exploration phase (oracle).

negative impacts. Our results show that across all profiles, the CF algorithm excels in producing the same ranking as the expected outcomes, for the worst configuration, denoted by *True* in the second column of Table 2.

Inaccurate Average Reward. Fig. 13 illustrates how the Reward Estimator module provides accurate and robust results even with insufficient data, demonstrating that averaging rewards from observational data fails to capture the nuanced estimates that CI provides. The figure demonstrates the impact of the number of samples per configuration on the performance (i.e., when the number of samples for each configuration, denoted by k, is less than 40).

For all configurations in each profile, we generate the GT reward, R_{GT}, as well as the CF estimated reward, R_{CF}, with the reward achieved from k samples, denoted by R_k. The R_{GT} (R_k) for each configuration is calculated using the average CT for up to 100 ($= k$) samples for each configuration. For each profile, we sort R_{GT}, R_{CF} and R_k and compare the GT reward ranking with both R_{CF} and R_k, using the *Ranking Score* function. This function provides a score based on the number of configurations that the R_{CF} and R_k ranks match the R_G ranks. For each profile, after sorting R_{GT}, R_{CF} and R_k, the *Ranking Score* function compares the R_G rankings with that of both R_{CF} and R_k results, and calculates the number of configurations that match between the CF estimates and GT reward as well as between the average reward of k samples and GT reward. This function provides the average number of configurations in both CF and the average reward of k

Profiles	Min r_{GT} = Min r_{CF}	R_{CF} RMSE
1	True	0.06
2	True	0.03
3	True	0.01
4	True	0.01

Table 2: Evaluating the CF algorithm results with GT, for the CI Limited Space module

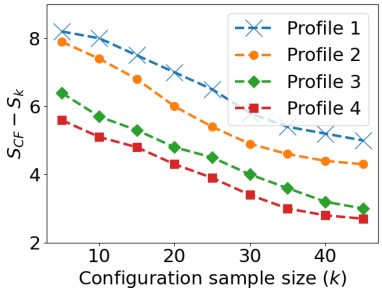


Figure 13: Comparing the results of CF algorithm and average reward of k samples with GT reward

samples with the same (or close enough) ranks as the GT reward. For evaluation purposes only, we consider a threshold value, denoted by $\varepsilon = 0.02$, to account for small variations (less than 2% difference from the GT reward) between the mismatched ranks. In this way, we can ignore the mismatch configurations in which the difference between the GT reward and the R_{CF} or R_k is less than ε. In other words, for each rank, if the following condition is valid for any of the average reward of k samples, the achieved R_k is assumed to be true and matched with the GT reward.

$$\frac{|R_{GT}[rank] - R_k[rank]|}{R_{GT}[rank]} \leq \varepsilon. \quad (2)$$

The condition (2) is written for the average reward of k samples. However, we similarly apply it to the CF results, by replacing R_k with R_{CF} in (2). The rank score for the CF estimates and average reward of k samples are denoted by S_{CF} and S_k, consecutively.

In Fig. 13, for $k = 5$ up to 45, we plot the difference between S_{CF} and S_k. This number for each value of k represents for how many more configurations on average the CF estimates are matched with the GT reward, compared to the average reward over k samples. By increasing the number of samples, the average reward gets closer to the GT reward, and therefore, more configurations match with the GT ranking.

6 Related Work

In networking, data-driven approaches for configuration tuning are highlighted for several scenarios [3, 11]. Several research studies consider TCP stack tuning [1, 2, 4, 15, 26, 29, 32,

33, 48]. Static protocol tuning [2, 15] fails to provide the best performance across fluctuating network conditions and sometimes may even result in performance degradation (compared to default configurations performance).

Reinforcement learning (RL) [26, 33, 47], supervised learning [4, 48], and multi-armed bandits [29] techniques are used to tune the configurations. The design of these approaches is very specific to the knobs; thus, it cannot be applied to jointly tune a different set of knobs [4, 32]. For example, TCP-RL [33] uses two separate RL models for two different configurations (initial *cwnd* and congestion control algorithm) to deal with scenarios of short and long flows. Only a few existing solutions support a wide range of configuration knobs with complex dependencies [29]. On the other hand, solutions that use live explorations of the configurations space suffer performance degradation for the new connections [3, 29]. Solutions cannot react agilely to these new connections due to the random trials that are applied to the first sessions. Additionally, the solution may continue the online exploration even after the bootstrapping phase. Also, it is challenging to collect firm network statistics on the server side in some cases (e.g., short flows) [33, 48]. However, none of the existing works dealt with the inaccurate rewards in their design.

On the other hand, contextual Gaussian Process (GP) [24, 39] is a well-established approach in black-box optimization. In this method, network conditions can be treated as direct contextual inputs to the GP. Our approach, however, takes a different perspective. We ensure that all user groups share a common context by dividing the user space into multiple groups based on profile features and network conditions. Therefore, we create multiple models to address contextual variations effectively. We believe that incorporating network conditions directly as inputs to the inference process can be challenging. Our approach only requires profile features to inference the best configurations, which allows us to handle the complexity more effectively.

7 Discussion

Configuration Knobs Selection. Config-Snob is a black-box solution that tunes a set of knobs. However, the selection of knobs is a challenging task [17]. Each system, application, and service has a wide range of knobs. The impact of knobs can differ from one use case to another. A deep analysis is needed to investigate the impact of knobs for each use case.

Client Privacy. As we pointed out that client information is used for profiling. Features like IP address, geolocation, and ISP information are learned during the connection setup. Generally, we need to ensure that clients' private information is collected with their permission and used safely without breaking user privacy [13, 46]. Config-snob emphasizes privacy by design. It ensures that only essential data required for system functionality is collected. Additionally, the stored data is abstracted into a profile guarantee that isn't tied to specific individuals. Furthermore, it adheres to data retention policies, deleting information after a specified period. We firmly believe users must be well-informed about data collection practices and provide them with options to opt out and customize privacy settings (e.g., geolocation). Another direction is to improve privacy by implementing profiling and inference on the client side (e.g., user APP). This approach is quick and can accurately collect the profile features but it has some challenges, such as overhead and model freshness.

Training Server Placement. Having both APP and Training servers on the same machine can be cost-effective, provide more user privacy protection, and be fast. However, the training server can serve different applications at the same time. Furthermore, a hierarchical approach can be used to deal with complex scenarios with multiple training servers (e.g., Pytheas framework [21])

Dynamic Updating Cycle. In the current deployment, we use a time-driven model updating. However, an event-driven approach can be used to boost the training process. For example, triggers can be used, such as the number of profiles that have been seen and the number of samples per profile.

Causal Inference. Considering online CI learning tools, instead of offline methods, can be beneficial for data-driven networking scenarios. This is more crucial for scenarios that have to deal with an insufficient amount of data, or if storing a large amount of data is not possible.

8 Conclusion

In this paper, we show that thoughtfully tuning the configurations for QUIC-based servers leads to significant performance improvement. We propose Config-Snob, a data-driven solution that utilizes the Bayesian Optimization approach and a causal inference algorithm to find the best configurations for the diverse network environment. Our solution can be applied to different networking protocol stacks (e.g., TCP, QUIC, HTTP, RTC, etc.) and services (e.g., web, video, etc.). Config-Snob fulfills all requirements of protocol tuning solutions including generality, adaptability, stability, scalability, low overhead, and reliability. In our future work, we aim to extend our design to tune the online parameters of the protocol stack. In addition, causal inference techniques show great potential to overcome some of the optimization tools issues. Therefore, further investigation and analysis are required.

References

[1] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Wanna make your tcp scheme great for cellular networks? let machines do it for you! IEEE Journal on Selected Areas in Communications, 39(1):265–279, 2020.

[2] Mohammad Al-Fares, Khaled Elmeleegy, Benjamin Reed, and Igor Gashinsky. Overclocking the yahoo! cdn for faster web page loads. In Proceedings of the 2011 acm sigcomm conference on internet measurement conference, pages 569–584, 2011.

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pages 469–482, 2017.

[4] Hamidreza Anvari and Paul Lu. Machine-learned recognition of network traffic for optimization through protocol selection. Computers, 10(6):76, 2021.

[5] Susan Athey and Stefan Wager. Estimating treatment effects with causal forests: An application. Observational Studies, 5:37–51, 01 2019.

[6] The GPyOpt authors. GPyOpt: A bayesian optimization framework in python, 2020. Accessed: 2022-07-20.

[7] Simon Bauer, Benedikt Jaeger, Fabian Helfert, Philippe Barias, and Georg Carle. On the evolution of internet flow characteristics. In Proceedings of the Applied Networking Research Workshop, pages 29–35, 2021.

[8] Theophilus A Benson. Illuminating the hidden challenges of data-driven cdns. In Proceedings of the 3rd Workshop on Machine Learning and Systems, pages 94–103, 2023.

[9] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. Queue, 14(5):20–53, 2016.

[10] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, and Van Jacobson. Bbr congestion control. IETF Draft draft-cardwell-iccrg-bbr-congestion-control-00, 2017.

[11] Jie Chuai, Zhitang Chen, Guochen Liu, Xueying Guo, Xiaoxiao Wang, Xin Liu, Chongming Zhu, and Feiyi Shen. A collaborative learning based approach for parameter configuration of cellular networks. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pages 1396–1404. IEEE, 2019.

[12] DB-IP. IP to Location + ISP database. https://db-ip.com/db/ip-to-location-isp, 2022. Accessed: 2022-07-20.

[13] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. arXiv preprint arXiv:1808.05096, 2018.

[14] Haiwei Dong, Ali Munir, Hanine Tout, and Yashar Ganjali. Next-generation data center network enabled by machine learning: Review, challenges, and opportunities. IEEE Access, 2021.

[15] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing tcp's initial congestion window. ACM SIGCOMM Computer Communication Review, 40(3):26–33, 2010.

[16] FCC. Measuring Fixed Broadband - Eleventh Report. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-eleventh-report, 2021. Accessed: 2022-07-20.

[17] Sishuai Gong, Usama Naseer, and Theophilus A Benson. Inspector gadget: A framework for inferring tcp congestion control algorithms and protocol configurations. In Network Traffic Measurement and Analysis Conference, 2020.

[18] Jenny Häggström, Emma Persson, Ingeborg Waernbaum, and Xavier de Luna. Covsel: An r package for covariate selection when estimating average causal effects. Journal of Statistical Software, 68(1):1–20, 2015.

[19] Kosuke Imai, In Song Kim, and Erik H. Wang. Matching methods for causal inference with time-series cross-sectional data. American Journal of Political Science, n/a(n/a).

[20] Iyengar, J., Swett, I. QUIC Loss Detection and Congestion Control. https://datatracker.ietf.org/doc/rfc9002/, 2021. Accessed: 2022-07-20.

[21] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pages 393–406, 2017.

[22] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols. In proceedings of the 2017 internet measurement conference, pages 290–303, 2017.

[23] Maggie Hei Greg Lewis Paul Oka Miruna Oprescu Vasilis Syrgkanis Keith Battocchi, Eleanor Dillon. EconML: A Python Package for ML-Based Heterogeneous Treatment Effects Estimation. https://github.com/py-why/EconML, 2019. Version 0.x.

[24] Andreas Krause and Cheng Ong. Contextual gaussian process bandit optimization. Advances in neural information processing systems, 24, 2011.

[25] Fan Li, Laine E Thomas, and Fan Li. Addressing Extreme Propensity Scores via the Overlap Weights. American Journal of Epidemiology, 188(1):250–257, 09 2018.

[26] Wei Li, Fan Zhou, Kaushik Roy Chowdhury, and Waleed Meleis. Qtcp: Adaptive congestion control with reinforcement learning. IEEE Transactions on Network Science and Engineering, 6(3):445–458, 2018.

[27] MaxMind. GeoIP2 City Database. https://www.maxmind.com/en/geoip2-city, 2022. Accessed: 2022-07-20.

[28] Usama Naseer and Theophilus Benson. Inspectorgadget: Inferring network protocol configuration for web services. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pages 1624–1629. IEEE, 2018.

[29] Usama Naseer and Theophilus A Benson. Configanator: A data-driven approach to improving {CDN} performance. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), pages 1135–1158, 2022.

[30] Marcin Nawrocki, Pouyan Fotouhi Tehrani, Raphael Hiesgen, Jonas Mücke, Thomas C Schmidt, and Matthias Wählisch. On the interplay between tls certificates and quic performance. In Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies, pages 204–213, 2022.

[31] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '15, page 417–429, USA, 2015. USENIX Association.

[32] Xiaohui Nie, Youjian Zhao, Guo Chen, Kaixin Sui, Yazheng Chen, Dan Pei, Miao Zhang, and Jiyang Zhang. Tcp wise: One initial congestion window is not enough. In 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC), pages 1–8. IEEE, 2017.

[33] Xiaohui Nie, Youjian Zhao, Zhihan Li, Guo Chen, Kaixin Sui, Jiyang Zhang, Zijie Ye, and Dan Pei. Dynamic tcp initial windows and congestion control schemes through reinforcement learning. IEEE Journal on Selected Areas in Communications, 37(6):1231–1247, 2019.

[34] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[35] OpenResty. OpenResty. Scalable Web Platform by Extending NGINX with Lua. https://openresty.org/en/, 2022. Accessed: 2022-07-20.

[36] scikit-learn. Clustering: K-means. https://scikit-learn.org/stable/modules/clustering.html#k-means, 2022. Accessed: 2022-07-20.

[37] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE, 104(1):148–175, 2015.

[38] Michael Stein. Large sample properties of simulations using latin hypercube sampling. Technometrics, 29(2):143–151, 1987.

[39] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. Safe exploration for optimization with gaussian processes. In International conference on machine learning, pages 997–1005. PMLR, 2015.

[40] Santiago Vargas, Gautham Gunapati, Anshul Gandhi, and Aruna Balasubramanian. Are mobiles ready for bbr? In Proceedings of the 22nd ACM Internet Measurement Conference, pages 551–559, 2022.

[41] Michael Welzl, Peyman Teymoori, Safiqul Islam, David Hutchison, and Stein Gjessing. Future internet congestion control: The diminishing feedback problem. IEEE Communications Magazine, 60(9):87–92, 2022.

[42] Gregory A Wilkin and Xiuzhen Huang. K-means clustering algorithms: implementation and comparison. In Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007), pages 133–136. IEEE, 2007.

[43] Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld. A performance perspective on web optimized protocol stacks: Tcp+ tls+ http/2 vs. quic. In Proceedings of the Applied Networking Research Workshop, pages 1–7, 2019.

[44] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein.

Pantheon: the training ground for internet congestion-control research. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 731–743, 2018.

[45] Liuyi Yao, Zhixuan Chu, Sheng Li, Yaliang Li, Jing Gao, and Aidong Zhang. A survey on causal inference. ACM Trans. Knowl. Discov. Data, 15(5), may 2021.

[46] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In NDSS, volume 62, page 66, 2012.

[47] Jia Zhang, Yixuan Zhang, Enhuan Dong, Yan Zhang, Shaorui Ren, Zili Meng, Mingwei Xu, Xiaotian Li, Zongzhi Hou, Zhicheng Yang, and Xiaoming Fu. Bridging the gap between QoE and QoS in congestion control: A large-scale mobile web service perspective. In 2023 USENIX Annual Technical Conference (USENIX ATC 23), pages 553–569, Boston, MA, July 2023. USENIX Association.

[48] Jianer Zhou, Xinyi Qiu, Zhenyu Li, Gareth Tyson, Qing Li, Jingpu Duan, and Yi Wang. Antelope: A framework for dynamic selection of congestion control algorithms. In 2021 IEEE 29th International Conference on Network Protocols (ICNP), pages 1–11. IEEE, 2021.

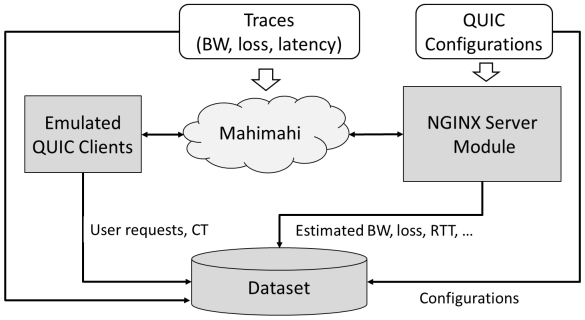


Figure 14: Oracle Experiment Setup.

Table 3: Used QUIC configuration knobs

Knobs	Values*
stream_init_data_pacing_bytes	**20KB**, 100KB, 300KB
stream_init_data_pacing_rate	20M, **90M**, 120M
stream_init_data_pacing _first_data_ack	**on**, off
buffered_data_threshold	**0**, 1M, 10M
ping_in_tail	on, **off**
retransmission_timeout_multiples	0, **0.25**, 1
loss_threshold_in_probe_up	**2%**, 10%, 50%
disable_condition_in_probe_up	on, **off**

*Default values are in **Bold**

A Oracle Description

Fig. 14 presents the data collection process. The system consists of three main parts: QUIC server, clients, and Mahimahi tunnel. Mahimahi is used to emulate different network scenarios. First, the Measuring Fixed Broadband report by FCC [16] is used which is summarized in Table 4. Additionally, the table shows the simulated network setup for the cellular networks [28, 43]. Second, real-world trace-driven network tunnels are simulated (Pantheon [44]). Last, we used traces that were collected from an online production server of the video-based service for three days. We replayed all the connections over the collected traces in video downloading granularity. The performance metrics, network statistics, and the used QUIC configurations are collected and stored in a dataset.

As we mentioned, we focus on tuning a set of configuration knobs listed in Table 3. Some of these knobs are continuous. Config-Snob uses BO models that support both discrete and continuous values. In the real deployment, continuous values of these knobs is used while a discrete number of values is used in the oracle. Thus, in the oracle, a certain number of configuration templates are created from all combinations of knobs' values. The number of used QUIC configuration combinations is 1944 templates. Table 3 shows the default and the used values for the configuration knobs. The default values in the table are selected just for the oracle testbed. While in real deployments, engineers set predefined configurations.

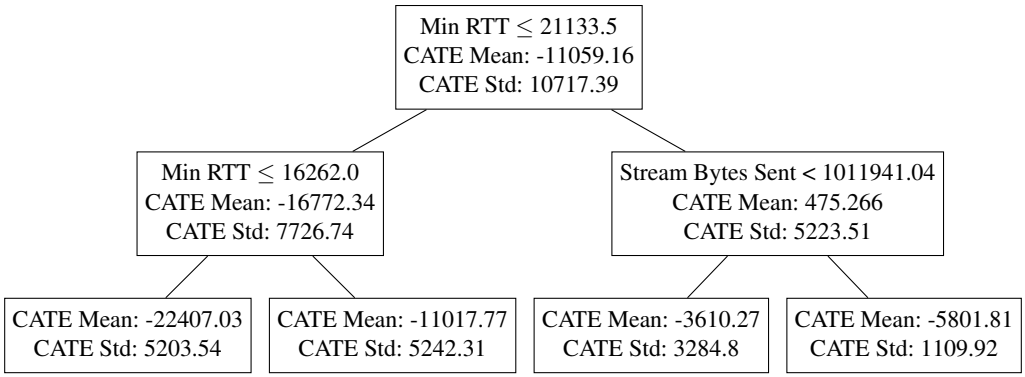


Figure 15: Clustering data samples using the trained CF tree.

Table 4: FCC+ Simulated Networks (Broadband and Cellular)

Clients Features			Mahimahi		
Interface	**Speed***	**Time**	**BW***	**Loss**	**RTT***
DSL	10	Offpeak	10	0.14%	38
DSL	10	Peak	10	0.25%	40
DSL	25	Peak	25	0.21%	31
DSL	25	Offpeak	25	0.1%	30
Cable	50	Peak	50	0.1%	21.5
Cable	50	Offpeak	50	0.11%	20
Cable	100	Peak	100	0.09%	23.5
Cable	100	Offpeak	100	0.07%	23
Fiber	100	Peak	100	0.04%	12.5
Fiber	100	Offpeak	100	0.04%	12.5
Fiber	500	Peak	500	0.01%	8
Fiber	500	Offpeak	500	0.02%	8
4G	10	Offpeak	10	0.1%	70
4G	10	Peak	10	0.15%	100
3G	3.54	Offpeak	3.54	0.1%	90
3G	3.54	Peak	3.54	0.15%	120

* Speed and bandwidth (BW) are in Mbps, RTT in milliseconds.

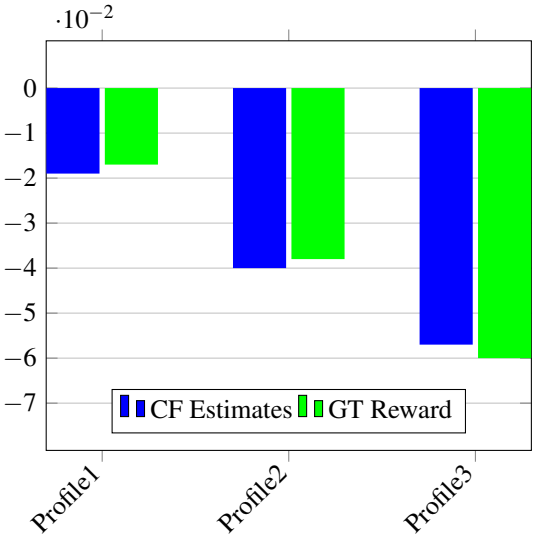


Figure 16: Evaluation of the CF results with the GT reward.

B CI Solution Choices Evaluation

Fig. 16 demonstrates the evaluation of the CATE generated by the CF algorithm, compared to the GT, for each profile. We consider three profiles $1-3$ with BW values of 25, 50 and 100 Mbps consecutively. The overall estimation accuracy for the CF algorithm for this scenario is 96.5%. We also provide the results of the clustered data samples using the CF tree approach (see Fig. 15), in which different samples are clustered based on their improvement ratios. The tree, demonstrated in Fig. 15, is generated using the selected features using the CovSel algorithm.