Hussein Darir*, Hussein Sibai, Chin-Yu Cheng, Nikita Borisov*, Geir Dullerud, and Sayan Mitra

MLEFlow: Learning from History to Improve Load Balancing in Tor

Abstract: Tor has millions of daily users seeking privacy while browsing the Internet. It has thousands of relays to route users' packets while anonymizing their sources and destinations. Users choose relays to forward their traffic according to probability distributions published by the *Tor authorities*. The authorities generate these probability distributions based on estimates of the capacities of the relays. They compute these estimates based on the bandwidths of probes sent to the relays. These estimates are necessary for better load balancing. Unfortunately, current methods fall short of providing accurate estimates leaving the network underutilized and its capacities unfairly distributed between the users' paths. We present *MLEFlow*, a maximum likelihood approach for estimating relay capacities for optimal load balancing in Tor. We show that *MLEFlow* generalizes a version of Tor capacity estimation, *TorFlow-P*, by making better use of measurement history. We prove that the mean of our estimate converges to a small interval around the actual capacities, while the variance converges to zero. We present two versions of *MLEFlow*: *MLEFlow-CF*, a closed-form approximation of the MLE and *MLEFlow-Q*, a discretization and iterative approximation of the MLE which can account for noisy observations. We demonstrate the practical benefits of *MLEFlow* by simulating it using a flow-based Python simulator of a full Tor network and packet-based Shadow simulation of a scaled down version. In our simulations *MLEFlow* provides significantly more accurate estimates, which result in improved user performance, with median download speeds increasing by 30%.

Keywords: Tor, capacity estimation, load balancing, maximum likelihood estimation, shadow simulator, privacy

DOI 10.2478/popets-2022-0005
Received 2021-05-31; revised 2021-09-15; accepted 2021-09-16.

***Corresponding Author: Hussein Darir, Hussein Sibai, Chin-Yu Cheng, Nikita Borisov, Geir Dullerud, Sayan Mitra:** All authors with the University of Illinois at Urbana-Champaign; {darir2,sibai2,ccheng32,nikita,dullerud,mitras}@illinois.edu.

1 Introduction

Tor [9] is a system for preserving online privacy and circumventing Internet censorship, with several million estimated daily users [29]. Tor operates by using a network of volunteer *relays* to forward an encrypted version of users' traffic, thus obscuring the source and/or the destination of network traffic. These relays have highly heterogeneous capacities, thus load balancing is essential to ensure consistent service to the users.

Load balancing is a well-studied topic; the security and privacy constraints of Tor, however, create some specific challenges for load balancing. First, for privacy reasons, Tor clients use source routing, where every client chooses which relays to use, precluding the use of a traditional feedback-based load-balancer. Instead, clients stochastically select relays based on a set of *weights*, corresponding to their network capacities. Second, trusting relays to report their own capacities, as was done in earlier versions of Tor, allows adversaries to compromise anonymity by lying about capacity to attract traffic to their nodes [3].

This motivated the development of *TorFlow*, a bandwidth monitoring system [22]. *TorFlow* uses external probes to monitor the performance of individual relays and uses this value to adjust the bandwidth value reported by the relay itself. The capacity estimates produced by *TorFlow*, however, vary considerably over time and between different *TorFlow* instances.

Our goal is to better understand the dynamics of bandwidth measurement and path allocation in Tor and design an improved measurement scheme. To this end, we developed a mathematical model of bandwidth measurement in the Tor network. This model makes a few simplifying assumptions but allows us to model the behavior of estimation algorithms. In particular, we find that an older version of *TorFlow* that was briefly deployed actually performs maximum likelihood estimation (MLE) of relay bandwidth according to our model. Furthermore, we derive bounds both on the mean and the variance of this estimate. We then propose a revised estimation mechanism that performs MLE using *multiple* observations, called *MLEFlow*; we derive both a closed-form and numerical approximations that can

be used to efficiently compute this estimate. We show that with multiple rounds of observations the variance of *MLEFlow* quickly drops, providing an increasingly accurate estimate of relays' capacities.

We then perform extensive simulations of both *TorFlow* and our MLE algorithm to validate the results of our analysis in more realistic settings. We use a custom-built flow-based simulator, written in Python, to simulate the behavior of the entire Tor network under both *TorFlow* and our proposed *MLEFlow*. We also use Shadow [15], a simulation framework that runs actual Tor code and simulates network events at a packet level; Shadow provides higher-fidelity simulation but can only be run on a scaled-down version of the Tor network.

In both cases, we confirm our analytical results, showing that *MLEFlow* results in much more accurate estimation of network capacities of relays, which, in turn, results in significantly better balancing of load for user traffic. When simulating simultaneous downloads, 75% of the users see improved download speeds, with a median improvement of 30%.

2 Path Allocation in Tor

The current Tor network consists of around 6000 *relays* [28] that are used to forward user traffic. To create a connection, a user chooses a *path* of three different relays to construct a circuit that forwards traffic in both directions. Only the user knows the entire path; the relays know only their predecessor and successor, obscuring the relationship between clients and destinations. The traffic is also encrypted / decrypted at each node to hide the correspondence between incoming and outgoing traffic from a network observer.

Relays in Tor have network capacity[1] sizes that differ by orders of magnitude (see Figure 3). This creates a need to balance the load between relays to better utilize the relays' capacity and to ensure that users do not encounter bottlenecks.

Relays also have different capabilities and can be divided into three classes: *exits*, which can be used in the last position of the path, *guards*, which can be used in the first or second position, and *middles* which can only be used in the second position [27]. We denote the corresponding sets of relays by E, G, and M, respectively. To create a path, relays are sampled from these sets with a probability proportional to their estimated capacity. For example, if we define $C[j]$ to be the estimated capacity of relay j, then the probability of choosing relay $j \in E$ as the last node in a path is $w^e[j] = C[j]/\left(\sum_{k\in E} C[k]\right)$; likewise for guard nodes being chosen in the first position. The middle position can be chosen from both guard and middle nodes; to balance bandwidth among classes, guard node capacity is adjusted by a multiplier W_{mg}; i.e., a guard node $j \in G$ is chosen for the middle position with probability:

$$w^m[j] = \frac{W_{mg}C[j]}{\sum_{k\in G} W_{mg}C[k] + \sum_{k\in M} C[k]}.$$

The multiplier is computed as:[2]:

$$W_{mg} = \frac{\sum_{k\in G} C[k] - \sum_{j\in M} C[j]}{2\sum_{k\in G} C[k]}.$$

It is easy to see that, in this scenario, if the estimated capacities are equal to the true relay capacities, which we will call $C^*[j]$, the expected number of paths using each exit relay will be proportional to its bandwidth; likewise, the expected number of paths using each guard and middle node will be proportional to their bandwidth. Using $X[j]$ to denote the number of paths on relay j, we have:

$$\mathbb{E}[X[j]]/C^*[j] = \mathbb{E}[X[k]]/C^*[k] \quad \text{for } j,k \in E$$
$$\mathbb{E}[X[j]]/C^*[j] = \mathbb{E}[X[k]]/C^*[k] \quad \text{for } j,k \in G \cup M$$

Thus, in expectation, each path would have the same bandwidth—$C^*[j]/\mathbb{E}[X[j]]$ for $j \in E$. Our goal is therefore to estimate these capacities as accurately as possible.[3]

2.1 Security Considerations

Since capacity estimation is used as input for path selection, it is intricately intertwined with security and

1 By "capacity" we refer to the smaller of upload and download bandwidth limit on the relay. This may be imposed by the ISP, the network configuration, or manually configured by the relay operator. In some cases, there may exist other bottlenecks on the path between two relays but a per-node bandwidth limit is a common and useful model of network capacity constraints.

2 This is a somewhat simplified presentation that describes the scenario where exit bandwidth is scarce and there is more guard bandwidth than middle bandwidth, as is the case in the actual Tor network. See the Tor Directory Specification for more details on how other cases would be handled [27].

3 Note that some research suggests allocation other than proportional to bandwidth results in better performance [13, 24]; nevertheless, an accurate capacity estimate is still needed for these alternative path allocation strategies.

privacy properties of Tor. If a user picks a path that consists of nodes under the control of an adversary, that adversary will have full visibility into the path and rendering Tor's anonymity protections entirely ineffective. In fact, with the use of timing analysis, it suffices to merely *observe* the network traffic at the first and last Tor relay [34] to link a user with a destination. Therefore, if capacity estimation gives the adversary's selected relays higher weight, this will increase the chance of anonymity compromise.[4]

A relay itself has the greatest visibility into its own network capacity, however, a compromised relay can simply lie and inflate its capacity without deploying high-bandwidth relays [3]. An alternative approach is to use probes to determine the network capacity. This is the approach we take in this paper, but it does not eliminate the possibility of attack. A relay may identify probes and treat them preferentially, rather than having them compete with regular traffic, thus inflating its estimate [30]. Alternately, an adversary may predict when a certain relay will be probed and perform denial-of-service on the relay, or the prober, to artificially reduce this estimate [17]. The latter attack can easily be defeated by randomization; the former is harder to defend, but we are optimistic that censorship circumvention research that aims to prevents identification of undesirable types of traffic can be brought to bear on this problem.

Some alternative approaches to probing, and their security tradeoffs, are discussed in Section 6

2.2 Tor Capacity Estimation

We next explain how capacity estimation is done in Tor. Each relay estimates its own network capacity by computing the maximum sustained download and upload bandwidth over a 5-second period over the last 5 days and reports this value to directory authorities, who then compile it across all relays and distribute the information to the clients in a *consensus* document, published every hour. The Tor directory specification [27] calls this value the *observed bandwidth* but we will refer to it as *self-reported bandwidth* to emphasize that it is supplied by the relay. We will use $b_t[j]$ to refer to the self-reported bandwidth of relay j in the consensus document published at time t.

This self-reported value is adjusted based on the results of a probe, which we will call the *measured bandwidth*, denoted by $m_t[j]$. Intuitively, if the consensus weights are properly set, the measured bandwidth of all relays should be roughly similar. The adjustment is computed by first calculating the average measured bandwidth across all relays, μ_t, and then multiplying the self-reported bandwidth by the ratio of the measured bandwidth and the average: $C^A_{t+1}[j] = b_t[j]m_t[j]/\mu_t$.

This version of capacity estimation has been in use in Tor since 2012, first implemented in *TorFlow* [22]. This method, however, has a number of disadvantages. A relay that is not sufficiently loaded may underestimate its self-reported bandwidth; this leads to a well-documented ramp-up period of new relays, where their low self-repoted bandwidth leads to a small estimated capacity and low load, which in turn leads to low self-reported bandwidth [6]. But even established relays see their self-reported bandwidth change. Figure 1 shows the self-reported bandwidth of 10 randomly selected relays over the month of May 2020, demonstrating significant variation over the period. Figure 1 also plots the ratio between the minimum and maximum self-reported bandwidth for all relays that were present for the entire month. The median such ratio is 20% and many relays have a ratio that is much higher. An additional problem is that the use of self-reported bandwidth makes it possible for a relay to influence its bandwidth. Johnson et al. [18] observe that a very large self-reported value is likely to result in a high consensus weight (although it may be conspicuous).

A previous configuration of *TorFlow* did not use self-reported bandwidth; instead, it adjusted the previous weight based on the current observation: $C^{TF}_{t+1}[j] = C^{TF}_t[j]m_t[j]/\mu_t[C]$.[5] The intuition is similar: if the current weight of the relay is too high, it will have a below-average performance, and its weight will be adjusted down, and vice versa. We will denote this version by *TorFlow-P* and we will study its properties in more detail in Section 3. We will note that Tor switched away from *TorFlow-P* because, when deployed, the feedback mechanism allowed the weights to significantly deviate

4 Simply running a high-capacity relay, or many moderate-capacity relays, can also be used to carry out this attack without tampering with capacity estimation (and has been in the past [7, 33]). There have been some proposals (e.g., [10, 24]) and several deployed strategies to increase the diversity of selected paths rather than selecting them solely according to bandwidth to make such attacks less likely.

5 This version of *TorFlow* actually was designed as a full-fledged PID controller, but its default configuration set the integral and derivative parameters to 0, and the proportional parameter to 1 [21].

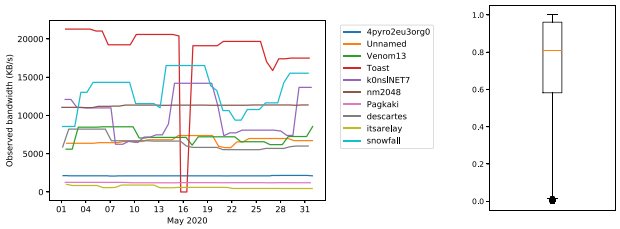


Fig. 1. Variation in self-reported bandwidth in Tor relays over the month of May 2020: plot of 10 randomly selected relays and a box plot of the ratio between the minimum and maximum self-reported bandwidth of all relays that were present for the entire month.

from network capacities, an issue we will discuss in Section 4.1.

A recent effort to upgrade and re-engineer *TorFlow* has resulted in the development of *sbws* [19]. *sbws* changes the way measurements are collected, but implements a version of the *TorFlow* scaling, which is intended to mimic *TorFlow* with only minor differences [26]. However, upon a review of *sbws* source code, we discovered an undocumented change in the way that weights are adjusted: *sbws* uses the minimum of the self-reported and the consensus weight: $C^S_{t+1}[j] = \min(C^S_t[j], b_t[j])m_t[j]/\mu_t$. This makes the *sbws* implementation a hybrid between *TorFlow* and *TorFlow-P*. It is still susceptible to underweighting relays with low self-reported bandwidth, but is more resilient to self-reported bandwidth that is too high. As of this writing, sbws and Torflow are each deployed on several Tor bandwidth authorities, with the consensus bandwidth being computed as the median of all authorities.

2.3 Simulation Frameworks and Baselines

The Shadow simulation framework [15] is a state-of-the-art discrete event simulator that is commonly used to study the Tor network. Shadow runs the actual implementation of Tor to emulate a number of relays running on a single host, communicating over a custom-built simulated network. Shadow has been widely used to analyze various properties of Tor as well as potential improvements.

Shadow cannot run Python, which is used to implement *TorFlow* and *sbws*. A C *TorFlow* plugin for Shadow has been previously developed, which mimics some of the behavior of *TorFlow*. However, it in effect implements *TorFlow-P* instead of *TorFlow* because the relay self-reported bandwidth is not reasonably available to the simulation: in the real-world, this value takes many days to stabilize [6], which is an impractical time interval to simulate in Shadow.

We have adapted the Shadow *TorFlow* plugin to simulate *TorFlow*, *TorFlow-P*, *sbws*, and *MLEFlow*. To better model *TorFlow* and *sbws*, we have used an idealized self-reported bandwidth, where the self-report is the actual (ground truth) bandwidth available to the relay. This is the value that the self-reported bandwidth is intended to capture, and relays frequently report values close to the actual available bandwidth, but at times fall short due to periods of low load (fig. 1). Thus our idealized simulations capture a "best-case" scenario for Torflow and sbws; to highlight this we denote our simulations as *sbws** and *TorFlow**. *MLEFlow* and *TorFlow-P* do not use the self-reported bandwidth and thus do not encounter this issue.

As Shadow is running the actual Tor implementation, it provides a highly realistic simulation, as has been validated in previous studies [14]. However, this causes it to be resource-intensive, and in practice it can only simulate a fraction of the Tor network. For example, our simulations of a Tor network scaled down to 3% of its actual number of relays runs approximately 36 times slower than real-time, despite using an 80-core 2.4,GHz Intel Xeon E7-8870 with 1 TB of RAM. To be able to simulate the entire network, we implement a simpler model of Tor, where each client stream is modeled as a flow and bandwidth is divided between flows using the max-min fairness, described in greater detail in Section 4. The flow-based simulator does not capture the intricacies of packet scheduling, flow and congestion control; as a result, it can simulate an entire Tor network with thousands of relays and up to millions of clients in Python on a desktop computer. Despite its simplifications, it captures essential behavior of Tor, as we show in appendix B by comparing its results to higher-fidelity Shadow simulations.

The flow-based model does not, however, well capture changes in load and ramp-up effects that cause self-reported bandwidth to vary over time; therefore, our flow-based simulations of *sbws* and *TorFlow* also use the idealized self-reported bandwidth.

2.4 Deployment

As discussed above, the *sbws* project was intended to re-engineer and modernize the bandwidth measurement and reporting mechanisms used in *TorFlow*, while keeping the consensus weight computation algorithm similar to *TorFlow*. Our goal with *MLEFlow* is complementary:

we intend to redesign the weight computation while reusing existing measurement and reporting infrastructure as implemented in *sbws*. The *sbws* implementation supports a configurable weight computation algorithm, termed *scaling* in its implementation and documentation; therefore, deployment of *MLEFlow* would require adding it as a new scaling algorithm in *sbws* and deploying it as a bandwidth authority.

For security and stability reasons, Tor directory authorities vote on a consensus document that combines data from multiple bandwidth authorities by taking the median reported value. As a first step, a single authority running *MLEFlow* could be added to the current mix of *sbws* and *TorFlow* authorities. The diversity of implementations may actually be a net security benefit, as an adversary would have to influence several different algorithms, making attacks more complex. To realize the full benefits of *MLEFlow*, however, most or all bandwidth authorities should be upgraded to the new algorithm. This would result in improved performance from better load-balancing, which in itself can lead to better anonymity [8], and it would remove the reliance on the easy-to-manipulate self-reported bandwidth of *TorFlow* and *sbws*. An additional deployment security consideration is the ramp-up period of new relays [6]: *MLEFlow* learns correct relay capacities much more quickly (see Section 4), making it easier to add new relays to the network, which could potentially help adversaries. Tor operators should examine whether the de facto probationary period for new relays created by *TorFlow*/*sbws* is a desirable tradeoff and implement explicit mechanisms for such a period if it is.

3 *MLEFlow*: Maximum Likelihood Estimation of Relays Capacities

We propose a new method *MLEFlow* for estimating the capacity of relays based on maximum likelihood estimation (MLE). To do this, we create a probabilistic model of the relationship between the actual relay capacities $C^*[j]$'s and the bandwidth measurements $m[j]$'s.

While this model simplifies some aspects of the operation of Tor, it allows us to induce a posterior probability on capacities based on the measurements and then apply MLE. Our analysis shows that *TorFlow-P* actually is equivalent to performing MLE in our model based on a single set of measurements per relay. Our framework *MLEFlow*, however, allows us to perform MLE based on *multiple* measurements of each relay over time; although this complicates the calculations, we are able to derive a closed-form approximation to the solution, which we name *MLEFlow-CF*; and a discretized computation that divides capacities into bins and uses these quantized capacities to iteratively finds the maximizer of the objective function, which we name *MLEFlow-Q*.

We show theoretically that both *TorFlow-P* and *MLEFlow-CF* converge to a value close to the actual capacities of the relays, but *MLEFlow-CF* has a variance that converges to 0 and is always lower than *TorFlow-P*. We confirm these results empirically, in addition to similar results for *MLEFlow-Q*, in Section 4. We also extend *MLEFlow-Q* to account for noisy measurements.

3.1 Simplified Model Description

We now introduce a simple model of the Tor network described in Section 2 for the purpose of analyzing *TorFlow-P* and *MLEFlow* theoretically. We assume the following:

1. Relays fall into a single category and each user path goes through only a single relay. This greatly simplifies our analysis, and yet gives useful results when users are bottlenecked at the exit relays, as is the case for the real Tor network (see Figure 3). Since exit bandwidth is scarce, and exits are only used in the final position, the bandwidth allocated to each three-hop path will be approximately equal to the scenario where each path is replaced by a one-hop path through its exit relay only. Thus our model should produce useful results for estimating exit bandwidth. Our simulations show that our model is also useful for estimating the capacity of guard and middle relays, albeit with larger estimation errors. However, as these relay classes have surplus bandwidth, estimation errors in these classes have a smaller impact on client performance.
2. A synchronized model where time is divided into epochs and user connections all terminate at the end of each epoch. During the t^{th} epoch, each client selects relays according to a weight vector w_t over all the relays. Since all relays can be used for single-relay paths, the distribution w_t over all the relays is published by the Tor authority instead of w_t^g, w_t^m, and w_t^e, over the three different categories. At the end of the epoch the weight vector is updated and the new vector is used by all users in the next epoch.

In reality, measurements are not synchronized with consensus periods; additionally, users may use a consensus document that is up to three hours out of date. However, as long as each relay is measured at most every 4 hours, at measurement time all users will be using a weight vector that reflects the estimate updated based on the previous measurement.

3. Users arrive to the network randomly following a Poisson process with rate λ_s, denoted by $\text{Pois}(\lambda_s)$. We denote the number of paths passing through the j^{th} relay in the i^{th} epoch by $x_i[j]$. Hence, given w_i, the number of paths using the j^{th} relay during the i^{th} epoch is a random variable $X_i[j]$ with distribution $\text{Pois}(\lambda_s w_i[j])$.

We will relax the single-relay path assumption in our experiments of Sections 4 and 5. There, we will use the realistic model described in Section 2 with three-relay user paths, while keeping the random Poisson arrival and the synchronized joining and leaving of the network. Formal proofs for the results given in this section can be found in Appendix A.

Given that the paths consist of single relays, the capacity of any relay is divided equally among all the paths passing through it. Formally, $m_i[j] = \frac{C^*[j]}{x_i[j]+1}$, where the added one in the denominator corresponds to the added test path. Hence, given w_i, the measurement of the j^{th} relay at the i^{th} iteration is a random variable $M_i[j] = \frac{C^*[j]}{X_i[j]+1}$. We add a normalization factor to the *TorFlow-P* update equation discussed in 2.2 to express it in terms of a probability weight vector:

Definition 1 (*TorFlow-P*)**.** *For any relay $j \in [n]$ and $t \in \mathbb{N}$, TorFlow-P updates the weight vector by:*

$$w_{t+1}^{TF}[j] = \frac{m_t[j]w_t[j]}{\sum_{k=1}^{n} m_t[k]w_t[k]}, \tag{1}$$

where we use the supersctipt TF in w_t^{TF} to identify TorFlow-P.

3.2 MLE Capacities Estimation

In this section, we will show how to compute MLE estimates of the relay capacities given a sequence of noisy measurements and weights published by the Tor authority. Maximum likelihood estimation is based on finding the parameter (here the relay capacity $C^*[j]$) that maximizes the probability of observing a certain measurement (here the bandwidths of the sequence of observation probes assigned to the j^{th} relay over $t+1$ periods $m_{[t]}[j]$). MLE is equivalent to Bayesian estimation with a uniform prior distribution on the parameters, i.e. each capacity in $\mathcal{C}$, the set of all possible capacities, is equally probable before the measurements are made.

More specifically, for any relay $j \in [n]$, the MLE estimate of its actual capacity $C^*[j]$ is the maximizer in $\mathcal{C} \subset \mathbb{R}^n_{\geq 0}$ of the probability of observing the full history of measurements $m_{[t]}[j]$, given the published weights $w_{[t]}[j]$ over the first $t+1$ periods. We add the superscript H to the capacity estimate to denote that the full history is considered. That is,

$$C_{t+1}^H[j] = \underset{\kappa \in \mathcal{C}}{\text{argmax}}\ f(\kappa, m_{[t]}[j], w_{[t]}[j]), \text{ where} \tag{2}$$

$$f(\kappa, m_{[t]}[j], w_{[t]}[j]) = \underset{X_{[t]}[j] \sim \text{Pois}(\lambda_s w_{[t]}[j])}{\Pr} \left(\frac{\kappa}{X_{[t]}[j]+1} = m_{[t]}[j] \right), \tag{3}$$

while all operations are done element-wise.

We now write the probability in equation (3) in terms of the known quantities: λ_s, $w_{[t]}$, and $m_{[t]}$. For any $j \in [n]$ and $t \in \mathbb{N}$ the function f of equation (3) can be written as follows:

$$f(\kappa, m_{[t]}[j], w_{[t]}[j]) = \prod_{i=0}^{t} \frac{e^{-\lambda_s w_i[j]}}{(\frac{\kappa}{m_i[j]} - 1)!} (\lambda_s w_i[j])^{\frac{\kappa}{m_i[j]} - 1}, \tag{4}$$

where f is only supported where the number of paths in each round $\frac{\kappa}{m_i[j]}$, is an integer.

Accordingly, we derive a closed form approximation of the maximization problem in (2) in the following theorem, for which the proof is in the appendix.

Theorem 1 (MLE closed form approximation)**.** *For any $j \in [n]$ and $t \in \mathbb{N}$, the MLE estimate of $C^*[j]$ given the weight and observation vectors $w_{[t]}[j]$ and $m_{[t]}[j]$ is*

$$C_{t+1}^H[j] \approx \exp\left(\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} \right), \tag{5}$$

where the approximation tends to equality as the user arrival rate λ_s gets large.

Definition 2 (*MLEFlow-CF*)**.** *For any $j \in [n]$ and $t \in \mathbb{N}$, our method MLEFlow-CF updates its weight in the t^{th} iteration by normalizing the MLE estimate $C_t^H[j]$ that uses the full history. Formally,*

$$w_t^{MF}[j] = \frac{C_t^H[j]}{\sum_{k=0}^{n} C_t^H[k]}. \tag{6}$$

3.3 *TorFlow-P* is an MLE Estimate

In this section, we show that *TorFlow-P* is the special case of *MLEFlow* in which the full history of measure-

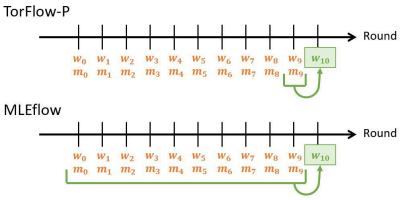


Fig. 2. *TorFlow-P* estimates relay capacity by MLE using only the most recent measurement; in contrast, *MLEFlow* uses the full history.

ments $o_{[t]}[j]$ and weight vectors $w_{[t]}[j]$ in equation (2) is replaced with *only* the most recent ones $m_t[j]$ and $w_t[j]$, respectively (see Figure 2). Below we will use superscript R to indicate the most recent value.

Using Theorem 1, the MLE estimate of $C^*[j]$ while considering only the last measurement would be:

$$C^R_{t+1}[j] = m_t[j]\lambda_s w_t[j]. \tag{7}$$

The normalized estimates in equation (7) would result in the same update as *TorFlow-P*:

$$w^{TF}_{t+1}[j] = \frac{m_t[j]w_t[j]}{\sum_{k=1}^{n} m_t[k]w_t[k]} = \frac{C^R_{t+1}[j]}{\sum_{k=1}^{n} C^R_{t+1}[k]}. \tag{8}$$

Thus, *TorFlow-P* is equivalent to estimating the capacities using ML considering only the last measurement and then normalizing to get the weight vector.

Fix any $j \in [n]$ and $t \in \mathbb{N}$. If *MLEFlow-CF* was used to generate the weight vectors over the first $t+1$ periods, we denote $C^H_{t+1}[j]$ by $C^{MF}_{t+1}[j]$. If *TorFlow-P* was used instead, we denote $C^R_{t+1}[j]$ by $C^{TF}_{t+1}[j]$.

3.4 Convergence of *TorFlow-P* and *MLEFlow-CF* Estimates

We show that, starting with any initial weight vector, the mean of the MLE estimates for any relay capacity, whether considering the full history in every update as in *MLEFlow* (Definition 2) or only the most recent measurement in every update as in Torflow (Definition 1), converges to a small interval around the actual relay capacity (Theorem 2).

Now, let's define the optimal distribution $w^*[j] = \frac{C[j]}{\sum_{k=1}^{n} C[k]}$, which we will use in the following theorem. Its proof is in the appendix. It relies on Taylor series expansion and Cauchy convergence test.

Theorem 2 (Estimates of both methods converge)**.** *For any $j \in [n]$, $t \in \mathbb{N}$, and a method $y \in \{TorFlow\text{-}P, MLEFlow\text{-}CF\}$,*

$$\mathbb{E}[C^y_t[j]] \le C^*[j]. \tag{9}$$

Moreover, as $t \to \infty$, $\mathbb{E}[C^y_t[j]] \ge C^*[j]\left(1 - \frac{1}{\lambda_s w^*[j]}\right)$. (10)

Corollary 3 (More users paths leads to a better convergence)**.** *As the rate of users arrival $\lambda_s \to \infty$, for any $j \in [n]$, $t \in \mathbb{N}$, and method $y \in \{TorFlow\text{-}P, MLEFlow\text{-}CF\}$, $\mathbb{E}[C^y_t[j]] \to C^*[j]$.*

Hence, from inequality (10), for the expected value of the estimate $C^{TF}_t[j]$ or $C^{MF}_t[j]$ of relay j to converge to a value within 20% of $C^*[j]$, the rate of arrival of new users to the network λ_s and the normalized actual capacity of the j^{th} relay $w^*[j]$ must satisfy $\lambda_s w^*[j] \ge 5$. This is the same as saying that the expected value of the number of paths allocated to the j^{th} relay by the optimal distribution w^* must be at least 5.

On the other hand, the relays that have very small capacities relative to the other relays, their $\lambda_s w^*[j]$ would be small. This would lead to larger error bound in (10). However, from (9), their expected estimated capacities would still be smaller than their actual ones. Hence, their entries in the probability distributions w_i would still be small.

Therefore, incoming users will be directed away from those relays for relays with higher capacities which we can estimate more accurately.

3.5 Variance of *TorFlow-P* > Variance of *MLEFlow-CF*

In this section, we show that the variance of the estimates generated by *MLEFlow-CF* is upper bounded by the estimates generated by *TorFlow-P* at any period. Furthermore, we show that the variance of our estimates converge to zero. This shows that our method *MLEFlow-CF* provides more stable and consistent estimates compared to *TorFlow-P*.

Theorem 4 (Variance of *TorFlow-P* > Variance of *MLEFlow-CF*)**.** *For λ_s big enough, for all $j \in [n]$, and starting with the same initial weights, $Var[C^{MF}_1[j]] = Var[C^{TF}_1[j]]$ and for $t > 1$,*

$$\frac{Var[C^{TF}_{t+1}[j]]}{Var[C^{MF}_{t+1}[j]]} > \frac{t+1}{\zeta}. \tag{11}$$

with $\zeta = 1 + \frac{1}{e^2} + \frac{2}{e}$.

Moreover, as $t \to \infty$, $Var[C_t^{MF}[j]] \to 0$.

The proof of the above theorems is in the appendix. It shows that the ratio of variance of *TorFlow-P* to that of *MLEFlow-CF* increases linearly with the number of epochs after the first one, at which they are equal. We further show that $Var[C_t^{TF}]$ is bounded, and it follows that $Var[C_t^{MF}]$ converges to zero.

Hence the variances of the MLE estimates when considering the full history of weights and measurements in *MLEFlow-CF* are close to the actual capacities more frequently than those of the estimates found using only the last measurement as in *TorFlow-P*.

We will show this experimentally in the next section.

3.6 Noisy Observations Model

In this section, we consider the case of noisy observations. We consider a noisy measurement model to capture network variations, whereby a measurement is randomly distributed around the mean $\frac{C^*[j]}{X_i[j]+1}$. We use a multiplicative noise model, i.e., the new observations are random variables of the form:

$$M_i[j] = \frac{C^*[j]Y_i[j]}{X_i[j]+1}, \tag{12}$$

where $Y_i[j] \sim \mathcal{N}_{[y_{\min}, y_{\max}]}(1, 1/20)$, a truncated normal distribution with mean one, standard deviation $1/20$, and truncated to be over the interval $[y_{\min}, y_{\max}]$. The interval $[y_{\min}, y_{\max}]$ can be tuned based on empirical observations (see Section 5).

3.7 *MLEFlow-Q*: Quanitzation-Based MLE

The MLE estimation of the capacity of relay $j \in [n]$ at time $t \in \mathbb{N}$, can be adjusted to account for the noise:

$$C_{t+1}^N[j] = \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}}\ g(\kappa, m_{[t]}[j], w_{[t]}[j]), \text{ where} \tag{13}$$

$$g(\kappa, m_{[t]}[j], w_{[t]}[j]) = \underset{\substack{Y_i[j]\sim\mathcal{N}_{[y_{\min}, y_{\max}]}(1,\frac{1}{20}),\\ X_{[t]}[j]\sim\text{Pois}(\lambda_s w_{[t]}[j])}}{\Pr}\left(\frac{\kappa Y_i[j]}{X_i[j]+1} = m_{[t]}[j]\right). \tag{14}$$

The number of paths passing through a relay is always finite. Also, the added noise is finite because of its truncated distribution. Given a measurement and the bounds on the noise, one can get the minimum and maximum numbers of paths that pass through the j^{th} relay. We solve the maximization problem in (14) by iterating over all possible numbers of paths, the support of the random variable $X_i[j]$, and computing the corresponding probability explicitly.

For any $j \in [n]$, $t \in \mathbb{N}$, an index $i \in [t]$, and $\kappa \in \mathcal{C}$, we define $x_{u,i}[j] = \lfloor \frac{\kappa y_u}{m_i[j]} - 1 \rfloor$ and $x_{l,i}[j] = \lceil \frac{\kappa y_l}{m_i[j]} - 1 \rceil$. Then,

$$g(\kappa, m_{[t]}[j], w_{[t]}[j]) = \sum_{i=0}^{t} \log\left[\sum_{x=x_{l,i}[j]}^{x_{u,i}[j]} \frac{1}{x!}(\lambda_s w_i[j])^x \exp\left(-\lambda_s w_i[j] - \frac{1}{2}\left(\frac{m_i[j](x+1)-\kappa}{\kappa\sigma_e}\right)^2\right)\right]. \tag{15}$$

With the introduction of noise in observations, the closed form solution of the MLE in *MLEFlow-CF* would not be accurate anymore. Hence, we introduce a third method, *MLEFlow-Q*, which discretizes the bounded capacity set $\mathcal{C}$ and iteratively find the maximizer of (15). This method can be applied to the non-noisy case as well by following the same steps to get an approximate maximizer of (4). Note that quantization requires knowing a lower and upper bound on the relay capacity, which can be estimated based on past observations.

MLEFlow-Q Implementation

Consider a partition $\bar{\mathcal{C}}$ of $\mathcal{C}$ into bins. The set $\bar{\mathcal{C}}$ contains the centers of the bins of $\mathcal{C}$. For any $j \in [n]$, $t \in \mathbb{N}$, and $\kappa \in \bar{\mathcal{C}}$, we define $L_t(j, \kappa)$ in the noiseless case to be:

$$-\lambda_s w_t[j] + \log\left(\frac{1}{\left(\frac{\kappa}{m_t[j]} - 1\right)!}(\lambda_s w_t[j])^{\frac{\kappa}{m_t[j]}-1}\right), \tag{16}$$

the t^{th} term of the sum when taking the log of (4), and in the noisy case to be the argument of the log in the t^{th} term of the sum in (15).

The sum of $L_t(j, \kappa)$ over measurement periods is stored in a variable $S_{t+1}(j, \kappa)$:

$$S_{t+1}(j, \kappa) = S_t(j, \kappa) + L_t(j, \kappa),$$

with $S_0(j, \kappa) = 0$. Then, the approximation of the MLE by *MLEFlow-Q* is computed by iteratively searching for the maximizer κ over the discretized capacity set $\bar{\mathcal{C}}$ in the following equation:

$$C_{t+1}^Q[j] := \max_{\kappa \in \bar{\mathcal{C}}} S_{t+1}(j, \kappa). \tag{17}$$

4 Flow-Based Simulation

To understand the properties and performance of the existing Tor capacity estimation and our proposed methods, we evaluated them using a flow-based simulation of the Tor network. These simulations model each flow in the network but do not capture details such as circuit construction, congestion, or flow control. The simulations are implemented in Python; see [4] for the algorithm implementation. We evaluate two versions of MLEFlow: *MLEFlow-CF* and *MLEFlow-Q*. We also evaluate *TorFlow-P* as well as the current Tor capacity estimation. As discussed in section 2.3, we use an idealized version of self-reported bandwidth when simulating sbws, denoting this by calling the simulation *sbws**.

We also evaluate the two cases *Actual* and *Quantized*, where actual capacities are given and quantized, respectively, and not estimated, for generating weight vectors for paths allocations to use their results as baselines to compare the results of other methods to.

We evaluate the performance of the different methods using two metrics: (a) the accuracy of the relay capacity estimates, and (b) the amount of bandwidth allocated to the user paths resulting from the weight vectors generated using the capacity estimates.

The simulation algorithm we have used is shown in Algorithm 1. The algorithm takes as input: the number of relays per path *pathsize* $\in \{1, 3\}$, the Poisson arrival rate of users λ_s (equals 10^6 in our simulations), the total number of measurement periods T to be simulated (equals 50 in our simulations), a *method* $\in$ {*Actual*, *Quantized*, *TorFlow-P*, *sbws**, *MLEFlow-CF*, *MLEFlow-Q*} to compute the capacities of the relays from measurements, an indicator *noisy* $\in \{0, 1\}$ for noisy observations, as described in Section 3.6, and lower and upper bounds on the multiplicative noise. In our simulations, we chose that noise to be distributed as $\mathcal{N}_{[0.7,1.3]}(1, \frac{1}{20})$ based on high fidelity simulations shown in Section 5. The algorithm outputs the bandwidth allocated to each user path and the weight vectors published over all periods between 0 and T.

The simulation algorithm iterates over measurement periods. In each period i, it generates the total number of users paths N_i that will join the network by sampling a Poisson distribution with rate λ_s in line 3. Then, it uses the weight vector w_i computed in the previous period as a probability distribution for the users to choose the relays of their paths from in line 4. In line 5, it uses the max-min fairness bandwidth allocation algorithm (corresponding to round-robin circuit scheduling [5]) to get the bandwidth allocated for each path, and thus generate the observation vector m_i. If *noisy* is true (or 1), then it multiplies m_i by a Gaussian noise clipped to be between y_{min} and y_{max} in line 6. After that, it computes w_{i+1} using the given *method* in line 7. Finally, it deletes all the paths for a fresh start of the next period.

Algorithm 1 Low fidelity simulation

1: **input:** *pathsize*, λ_s, T, *method* $\in$ {*Actual*, *Quantized*, *TorFlow-P*, *MLEFlow-CF*, *MLEFlow-Q*}, *noisy* $\in \{0, 1\}$, $y_{\min}, y_{\max}, w_0$.
2: **for** $i \in [0, ..., T]$ **do**
3: Pick the number of users $N_i \sim Poi(\lambda_s)$.
4: Construct users paths of *pathsize* relays using w_i.
5: Compute m_i using max-min bandwidth alloc.
6: Multiply m_i by $Y_i \sim \mathcal{N}_{[y_{\min}, y_{\max}]}(1, \frac{1}{20})$ if *noisy*.
7: Compute w_{i+1} based on m_i and w_i using *method*.
8: Delete all paths in the network.
9: **return:** $m_{0:T}, w_{0:T}$

We consider a network analogous to the current Tor network with 6481 relays as of April 2021. The relays are distributed as follows: 2733 are guard relays, 2570 are middle relays, and 1178 are exit relays (this includes any relays that have both the Exit and Guard flags set). Lacking a ground truth, we used the highest reported observed bandwidth over a period of a month to as the actual capacity of the relay in our simulation, similar to Traudt et al. [31]. The maximum capacity of all relays was 122 601 KB/s. Hence, the capacity set is $\mathcal{C} = [0, 122\,601]$. The distributions of the relays' capacities are shown in Figure 3.

To use *MLEFlow-Q*, whether for the noiseless or noisy observations case, we need to partition $\mathcal{C}$ into bins. From Figure 3, we can see that the relay capacities roughly follow a truncated exponential distribution. Thus, we choose the bins to be intervals of the form $[a^{b-1}, a^b]$ where a is a strictly positive real number and $b \in [1, ..., b_{max}]$ where $b_{max} = \lceil \frac{\log(\max(C[j]))}{\log(a)} \rceil$ for $j \in [n]$.

Detailed numerical results of the simulation are reported in Table 1 in the Appendix.

Discretization of $\mathcal{C}$ does not hurt performance. We tried different discretization resolutions of $\mathcal{C}$ and we chose bins with $a = 1.1$ as it results in a low quantization error and relatively small number of bins, $b_{max} = 196$. The maximum resulting quantization error is less than 5%.

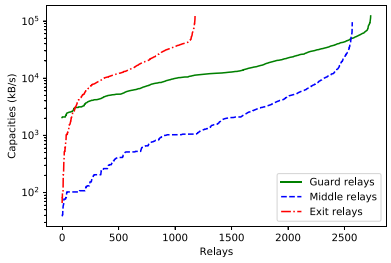


Fig. 3. Relays capacity distribution.

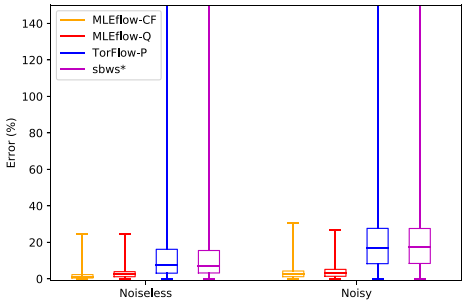


Fig. 4. Estimation error distribution in a single relay paths network after the 50^{th} measurement period in the noiseless and noisy observations scenario. For clarity the plot cuts off the maximum value for *TorFlow-P*, which is 288% in the noiseless case and 381% in the noisy case; and the maximum value for *sbws**, which is 497% in the noiseless case and 865% in the noisy case.

Moreover, using the quantized capacities to generate the weight vector instead of the actual capacities has a minimal impact on the balance of user paths: using the true capacities have a mean of 22.41 KB/s and a standard deviation of 0.77 KB/s, whereas using quantized values the paths have an identical mean and only a standard deviation of 0.99 KB/s.

Our simulations support our theoretical analysis in Section 3 for single relays paths. We tested the same network of Figure 3, but with users paths consisting of single relays, to validate our theoretical derivations in Section 3. The results are shown in Figure 4 and the grey columns in Table 1 (Appendix C). All three methods *TorFlow-P*, *MLEFlow-CF*, and *MLEFlow-Q*'s average estimation errors converge to zero. The average estimation error stayed below 5% for *MLEFlow-CF* and *MLEFlow-Q*, while it was higher for *TorFlow-P* and *sbws** at around 22% when the observations are noisy. Moreover, the standard deviation of the estimation error was at most 4.75% for our methods while it was up to 21% for *TorFlow-P*. This reflects that the variance of the capacity estimates of *MLEFlow-CF* and *MLEFlow-Q* are much smaller than those of *TorFlow-P*, in accordance with our theoretical analysis.

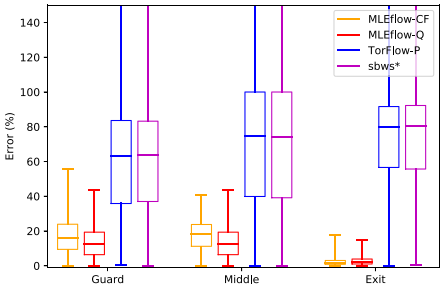


Fig. 5. Estimation error distribution in a three relays paths network for each type of relay after the 50^{th} measurement period in a noisy observations scenario. The maximum values for *TorFlow-P* and *sbws** are cropped for clarity; they range between 200% and 1150%

Our methods' advantages extend to the three-relays paths scenario. Figure 5 shows the estimation error distribution when users create paths of three relays rather than one. Both of our methods *MLEFlow-CF* and *MLEFlow-Q* preserved the average capacities' estimation error below 5%, while *TorFlow-P* and *sbws** increased significantly to around 82.17% for the Exit relays. For guard relays, the error increased to around 17% for our methods versus increasing to around 70% for *TorFlow-P* and 68% for *sbws**. Similarly, for middle relays, the estimation error increased to around 18% in our methods versus up to 84% for *TorFlow-P* and *sbws**.

Exit relays are the relays expected to be the bottlenecks of the users paths since they have the smallest total capacity of all three classes (see Figure 3) and each user path must use an exit relay. Being a bottleneck relay for a path, that relay determines the bandwidth allocated for the path in the network. If a relay is a bottleneck for most of the paths passing through it, it would allocate bandwidths to these paths in a similar way to the case where these paths were a single-relay ones which was described in Section 3. That means that we expect that our theoretical derivations would better extend to this type of relays first, more than the guard and middle counterparts. This can be seen by comparing the capacities estimation error statistics for guard and middle relays versus the exit ones in Figure 5. There is a noticeable increase in capacity estimation error of guard and middle relays compared to the exit ones. Fortunately, because exit relays are the usual bottlenecks, estimation errors in the capacity of middle and guard relays have a smaller impact on the user performance. This is supported by results in Figure 6: the bandwidth allocation to paths when using the actual (ground truth) capacity is very similar to that achieved

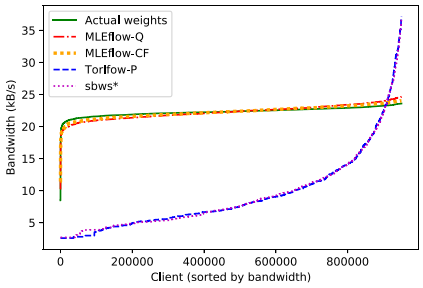


Fig. 6. Distribution of bandwidth allocated to paths using different estimation algorithms in a noisy observations scenario. We cut off the graph at the 95th percentile since the highest bandwidth paths in *TorFlow-P* are allocated over 30 000 KB/s.

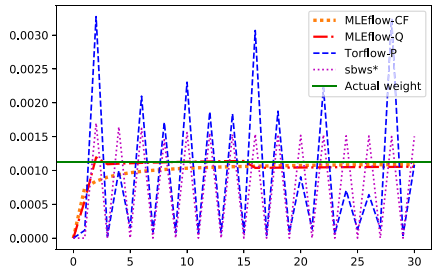


Fig. 7. The normalized estimated capacity of an exit relay joining the network after 15 measurement periods and staying for 30 periods, estimated using *TorFlow-P* and *MLEFlow-Q*.

by *MLEFlow-CF* and *MLEFlow-Q*, whereas *TorFlow-P* and *sbws** have much wider variation in the bandwidth allocated to user paths due to larger estimation errors.

MLEFlow-CF* and *MLEFlow-Q* give better and fairer bandwidth allocation than *TorFlow-P* and *sbws*. The means of the bandwidths allocated for paths using *MLEFlow-CF* and *MLEFlow-Q* are equal to that of the *Actual* scenario, while that of *TorFlow-P* and *sbws** are slightly smaller. The bigger advantage of our methods is the small range of bandwidth allocation. The maximum standard deviation is 1.23 while that of *Actual* is around 0.77. In contrast, *TorFlow-P* has a maximum standard deviation of around 175 and that of *sbws** is around 176, orders of magnitude larger than that of our method. Moreover, the maximum and minimum bandwidths allocated of our methods are similar to that of *Actual* while *TorFlow-P* and *sbws** had orders of magnitude larger maximum. That means that our methods distribute bandwidths more fairly than *TorFlow-P* and *sbws**, making the experience of using Tor more predictable. This can also be seen in Figure 6 for the 50^{th} measurement period. The bandwidths allocated for paths when using MLE estimation methods overlap those of the case when the actual capacities are used. They are flat curves resembling fair distribution of bandwidths versus the exponentially shaped distribution resulting from using *TorFlow-P* and *sbws**. This holds for both the noiseless and noisy observations cases.

Having a wrong estimate of the rate of the users arrival λ_s does not affect the quality of the estimates of *MLEFlow-CF* and *MLEFlow-Q*. We used an actual λ_s to generate users paths in our simulations in line 3 of Algorithm 1, while we used a different one in updating our weights in line 7. The results shown in Table 2 (see Appendix C), when compared to those in Table 1, show that the results are almost intact with reasonable deviations from the actual rate.

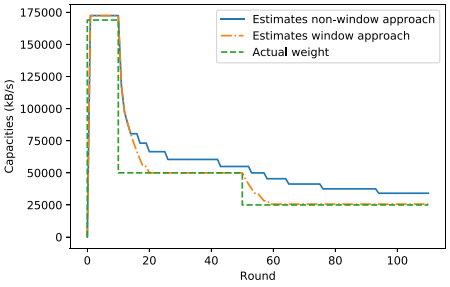


Fig. 8. The estimated capacity of a relay changing capacity using *MLEFlow* with windowed and non-windowed approach.

MLEFlow* handles new relays better than *TorFlow-P* and *sbws*. We show the result of estimating the capacity of a relay joining the network 15 iterations after we started estimating the capacities of all existing relays at the time. The results are shown in Figure 7. *MLEFlow* estimate converged to the actual capacity in couple of epochs while *TorFlow-P* and *sbws** stayed having large errors, even after 30 measurement epochs.

***MLEFlow* estimation accuracy is not affected when relays change capacities.** We evaluate the effect of relays changing capacities on the estimates we get using *MLEFlow*. The results are shown in Figure 8. As can be seen in the figure, *MLEFlow* can track the change in the capacity of a relay. We also test the idea of using a window of previous observations and not the whole history of observations as input to the *MLEFlow* algorithm. We simulate *MLEFlow* using the observations of the last 10 epochs only. As can be seen in the figure Figure 8, the number of epochs needed in order to converge to the new capacity is reduced significantly when using the windowed version of the algorithm.

4.1 Simulations of Underloaded Networks

Our theoretical analysis, as well as simulations in this section, assume that clients can utilize arbitrary amounts of bandwidth and are only bottlenecked on the Tor network. In practice, at times the client demand on Tor is lower than the overall available bandwidth, which changes the assumptions behind TorFlow and *MLEFlow*. Let us first consider *TorFlow-P*. Under high load, if the consensus weights are set in direct proportion to the relay capacities, the realized probe bandwidth at each node is expected to be the same. When load is low, however, this is no longer the case; proportional weighting results in each relay having a similar percentage of spare capacity, a large fraction of which can all be dedicated to the probe. Thus the probe bandwidth will be proportional to the relay capacity, and the algorithm will try to correct this imbalance. Relays that have above-average bandwidth will see their weights adjusted upwards, and other relays will be adjusted downwards.

This behavior was observed in the Tor network when *TorFlow-P* was deployed in 2011. High-bandwidth relays were receiving an increasingly large proportion of the weight, whereas low-bandwidth ones would see their weight drop, concentrating traffic on a smaller fraction of relays. Note that this is actually a sensible strategy for optimizing the available bandwidth of new circuits: the spare capacity of high-bandwidth nodes allows new circuits to realize performance that is simply impossible with nodes that are significantly below average.[6] However, such concentration of traffic at the fast relays raised concerns about potential compromise of those relays, which motivated the switch to adjusted bandwidth in Tor. It is notable that the latter approach also suffers in low-load conditions, as an underutilized relay may observe a bandwidth that is lower than its capacity (see fig. 1).

MLEFlow will likewise tend to misestimate capacity in an low-load scenario. In our simulations, the combined guard and middle capacity was nearly 4x higher than the exit capacity, resulting in lower utilization of guards and exits. Consequently, we can see in fig. 5 that estimation errors for these relays are significantly higher than for exits (though much lower than for *TorFlow-P*). However, we wanted to understand how the estimation would behave if the entire network was underloaded.

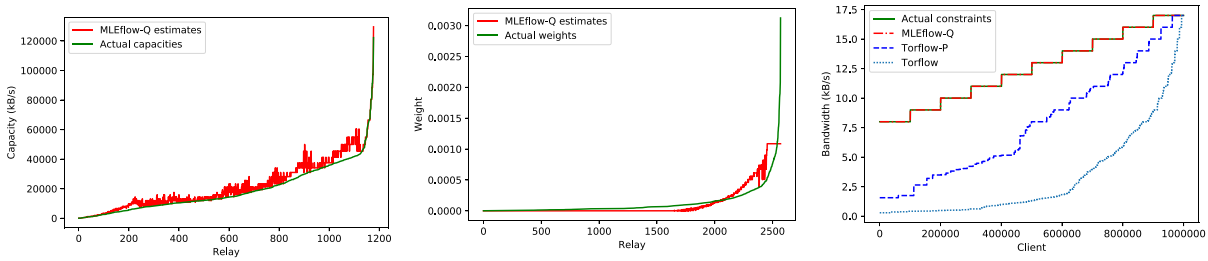
To simulate this, we adjusted simulation to add a bandwidth cap to each client flow, selected uniformly at random from the interval [8,18] KB/s. Since the average bandwidth of flows in the full utilization scenario was approximately 22 KB/s, the cap means that the flows can utilize at most about 60% of the Tor network capacity. We simulated *MLEFlow*, *TorFlow-P*, and TorFlow with adjusted observed capacity.

As expected, with an underloaded network, *MLEFlow* significantly overestimated the capacity. In fact, with quantized estimates, a large number of relays got classified into the largest bin. However, when we adjusted the bins to extend the upper limit of estimates, we found that the relative capacity estimate for exit nodes was reasonably accurate (fig. 9a). The estimates for middle relays were significantly worse, and we can see that low-bandwidth relays are assigned very small weights (fig. 9b). Notably, we do not see the same effect, concentrating weight among high-bandwidth nodes, among guard nodes, likely because guards have a high minimum required bandwidth. We observe that the estimation noise does not impact the performance of circuits; as can be seen in fig. 9c, nearly all circuits are able to realize their full capped bandwidth. *TorFlow-P* and TorFlow perform significantly worse in this scenario; the imbalance observed in our previous simulations, combined with client bandwidth caps, result in much worse utilization of the Tor network. An average circuit achieves only 50% of its bandwidth cap using *TorFlow-P*, and 18% using TorFlow.

5 High-Fidelity Packet-Based Simulations

We compare the performance of *MLEFlow-Q*, *TorFlow-P* and *sbws** in a simulated Tor network using Shadow [15], a high-fidelity event driven simulator for Tor. Shadow runs the actual C implementation of Tor relays, combined with a simulated network; to make the simulation manageable, Shadow uses a scaled down version of the Tor network that samples a fraction of relays and simulated clients. In our case, we configured a 1% and a 3% network. The 1% network contains 65 Tor relays: 21 guards, 36 middle relays, and 8 exits. The total bandwidth of the relays is 417 MB/s, split into 238 MB/s for the guards, 82 MB/s for the middle relays, and 97 MB/s for the exits. We simulate 500 clients, 3

6 Indeed, the average network performance during this time did not suffer and may have even increased, see https://metrics.torproject.org/torperf.html?start=2011-12-01&end=2011-12-31&server=public&filesize=1mb.



(a) Exit relay estimation in a low-load network.

(b) Middle relay estimation in a low-load network.

(c) Circuit bandwidth in a low-load network using MLEFlow, *TorFlow-P*, and TorFlow.

Fig. 9. Simulation results in a low-load network, where client capacities are capped to [8,18] KB/s, and as a result only 60% of the exit capacity can be utilized.

directory authorities, and a bandwidth authority. The 3% network contains 196 Tor relays (61 guards, 109 middle relays, and 26 exits). The total throughput is 1.3 GB/s (guards: 713 MB/s, middles: 252 MB/s, exits: 359 MB/s). We simulate 2000 clients, 3 directory authorities, and a bandwidth authority. Each client maintains a single file download stream during a consensus round. All streams are dropped and restarted at a new consensus round to make sure that the downloads utilize circuits generated by the latest consensus. We've also set the duration of each consensus round to 10 minutes to reduce the time needed to simulate each round; in real-life Tor, directory authorities generate a new consensus every hour. Clients also reuse consensus documents for up to 3 rounds. However, bandwidth authorities in the real world have a much longer refresh period. It can take as long as several days to complete a scan of the entire Tor network.

For *MLEFlow-Q*, we added functionalities of the algorithm to the Tor plugin run by the directory authorities. At each consensus round, the modified directory authorities read the measured observation of each relay from the prober (bandwidth authority) and execute the *MLEFlow-Q* algorithm to generate published bandwidth values for all relays. Each directory authority also maintains maximum likelihood score vectors to keep track of the published bandwidth history for each relay. The matrix is updated with the scores generated from the latest observations at each round.

To support *TorFlow-P*, we used the latest TorFlow implementation in the Shadow Tor plugin. The plugin reimplements the logic of TorFlow in C, as Shadow does not support running Python code used to implement TorFlow as well as the newer sbws system. As discussed in 2.3, Shadow does not run long enough to produce useful self-reported bandwidth. The existing TorFlow plugin uses the consensus weights instead, effectively implementing *TorFlow-P*. We extended the plugin to also implement TorFlow and *sbws** while using the actual, rather than self-reported capacities, just as in the flow-based Python simulations.

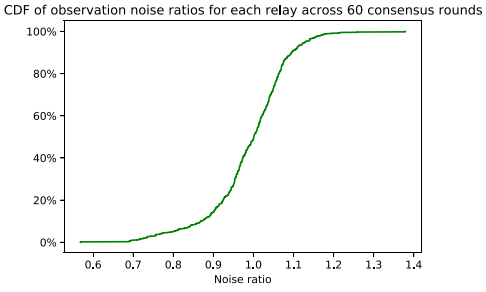
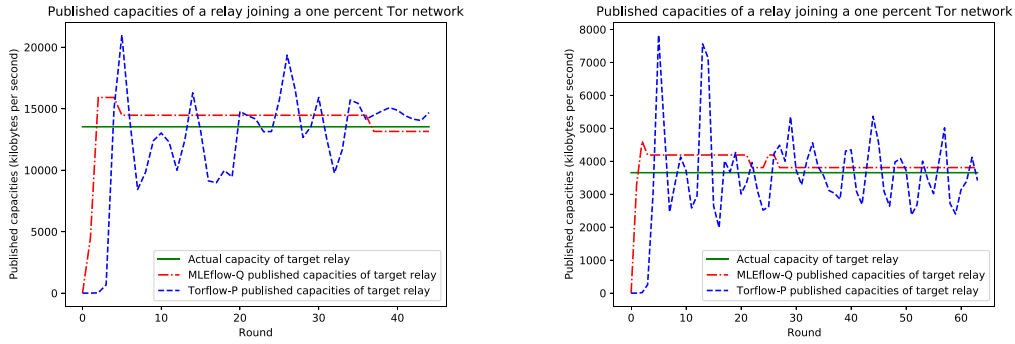


Fig. 10. CDF of noise ratios calculated as the ratio between the bandwidth of relay j observed in simulation, $o[j]$ and the modeled bandwidth—$C^*[j]/(1 + X[j])$, where $X[j]$ is the actual number of paths using relay j and $C^*[j]$ is the true capacity of relay j.

Estimating the noise ratio in the high fidelity simulation. The noise-free model we use in our bandwidth estimation assumes that the bandwidth of a relay is split evenly between all the circuits going through it (including the observation circuit). In practice, this will not be the case because some of the circuits may be bottlenecked at other relays; furthermore, congestion and flow control mechanisms in Tor and TCP will cause occasional imbalance between flows. To understand this effect, we ran a simulation of the 1% network and compared the bandwidth of the observation circuit to the bandwidth predicted by the model, i.e., the actual capacity of the relay divided by the number of circuits using the relay, $C^*[j]/(1 + X_i[j])$. We call the ratio between these two values the *noise ratio*. Figure 10 shows the maximum and minimum noise ratio of all relays in each round. We can see that the noise ratio is bounded between 0.7 and 1.3 for all rounds. These numbers are used to determine the plausible range of noisy measurements for *MLEFlow-Q*.

***MLEFlow-Q* handles newly joining relays better than *TorFlow-P*.** Our first analysis evaluates



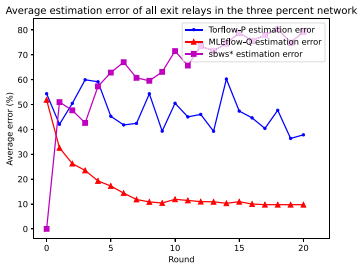
(a) 13 MB/s **(b)** 3.6 MB/s

Fig. 11. Published capacities of a previously unseen relay joining the 1% network. The target relay joins the network after 10 consensus rounds have already been simulated.

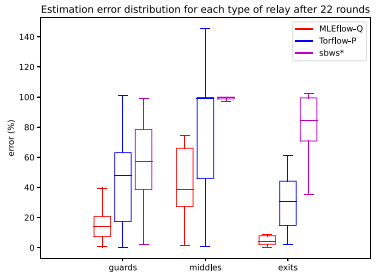
the performance of each algorithm when a new relay joins an already-converged network. We first run both *TorFlow-P* and *MLEFlow-Q* for 10 rounds (6000 seconds), after which a previously unseen relay joins the network. The new relay, whose true capacity is 13 MB/s, first start with zero published weight. The subsequent estimates generated by both algorithms are shown in figure 11a. For *MLEFlow-Q*, we observe that the estimation is stable and within a 10% error after 5 consensus rounds, with the estimation inching even closer to the true capacity after 40 rounds. Meanwhile, *TorFlow-P*'s estimates have larger errors throughout the simulation and shows significant fluctuations compared to *MLEFlow-Q*. We also ran the same experiment to estimate a previously unseen relay with a smaller capacity at 3.6 MB/s. We see similar results with this experiment where *MLEFlow-Q* manages to maintain a stable estimate with 5% error. Meanwhile *TorFlow-P*'s fluctuates around the true capacity (figure 11b).

***MLEFlow-Q* achieve a lower average estimation error than *TorFlow-P* with a narrower range of error.** Our second analysis looks at how each algorithm performs when estimating all relays with zero initial information. That is, all relays are assumed to have equal bandwidth in the beginning. We calculate the average estimation error of all exit relays in a whole network estimation run. The results are shown in figure 12a. We can see that *MLEFlow-Q* outperforms *TorFlow-P* by maintaining a low average estimation error around 10% after running for 10 consensus rounds. *TorFlow-P*, meanwhile, is only able to maintain an average error between 30% and 50%. The minimum achieved error by *MLEFlow-Q* is 8%, which is nearly optimal considering that quantization can introduce an error of up to 10%. We see similar results in the 3% network, where *MLEFlow-Q* achieved a sub 10% average estimation error after 20 rounds compared to *TorFlow-P*'s higher estimation error. Moreover, *MLEFlow-Q* outperforms *sbws** for which the average estimation error of the exit relays is above 50%. We aggregated the estimation error distribution for all categories of relays, shown in Figure 12b. In all cases, *MLEFlow-Q* achieved a lower average error with a narrower range of error. The appendix contains more plots of final estimation results and a comparison of estimation using flow- and packet-based simulations.

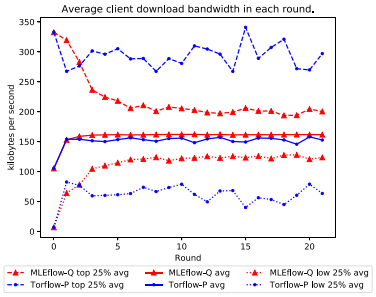
***MLEFlow-Q* estimates give fairer bandwidth allocation than *TorFlow-P* and *sbws** estimates.** Finally, we analyze the effects of *MLEFlow-Q* and *TorFlow-P* on the bandwidth distribution across the entire network. Figure 12e shows the distribution of download speed across the 2000 clients in the simulated 3% network using consensus weights generated by *MLEFlow-Q*, *TorFlow-P* and *sbws** after 22 consensus rounds. The figure shows the effects that each algorithm has on the load balancing across the entire network. Around 75% of all clients under *MLEFlow-Q* achieved a higher bandwidth compared to *TorFlow-P* and *sbws**. *MLEFlow-Q* resulted in a more even bandwidth distribution with a higher average bandwidth at 162 KB/s compared to *TorFlow-P*'s 153 KB/s and *sbws**'s 139 KB/s. The load balancing effect can also be seen in the round-to-round analysis shown in figures 12c and 12d. For *MLEFlow-Q*, we see a smaller average bandwidth for the top performing clients, but a larger overall average and a better average for the worst performing clients compared to *TorFlow-P* in all rounds. The standard deviation of bandwidths in *MLEFlow-Q* is also much lower than *TorFlow-P* in all rounds, which implies a fairer bandwidth allocation when using *MLEFlow-Q*.



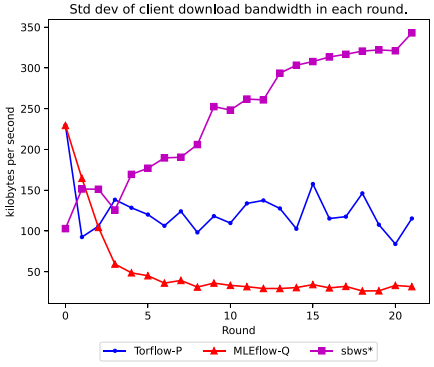
(a) Comparing average estimation errors of all exit relays in each round.



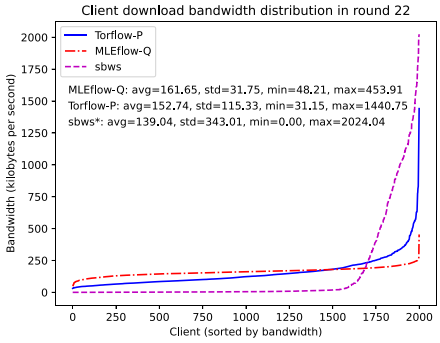
(b) Estimation error distributions for each type of relay.



(c) Aggregate mean progression of client download bandwidth distribution in each consensus round.



(d) Aggregate std deviation progression of client download bandwidth distribution in each consensus round.



(e) Client download bandwidth distribution after bandwidth estimation has converged.

Fig. 12. Results are obtained after 22 consensus rounds in the 3% network using Shadow.

5.1 Measurement Bandwidth

MLEFlow-Q, like TorFlow, relies on active probes to measure bandwidth. This imposes resource costs on the scanner, and, to a lesser extent, the relays being measured. For example, the documentation of sbws, the successor of TorFlow, suggests that 12–15 GB/day bandwidth will be used during scanning [25]. To understand this better, we decided to measure the amount of bandwidth used by the scanner. Note that TorFlow and sbws use a fixed limited number of parallel scanners, whereas in our simulations our bandwidth authority simultaneously scans all the relays. We found that our scanner used about 300 MB/s to do this in our simulations.

This figure can be used to characterize the tradeoff between scanner bandwidth utilization and scanning speed. A scanner can partition the network into a set of shards, and scan each shard individually. For example, our 3% network simulations suggest that splitting the network into 33 shards of 3% each would require 300 MB/s at the scanner. A scanner with lower capacity can split the network into more shards; e.g., a common 1 Gbit uplink can be used to scan shards representing 1% of the network. Conversely, a scanner with very high bandwidth could conceivably scan the entire network at once; using our flow-level Python simulations, we estimate that a simultaneous scan of the entire network would require 14 GB/s.

We performed simulations to verify that bandwidth-limited scanners can still produce accurate results. We configured a Shadow simulation of a 1% network with a scanner limited to 100 MB/s (achievable with a 1 Gbps network connection) and found no significant difference in estimation errors. We expect that bandwidth savings can also come from limiting the bandwidth of each measurement flow; this will underestimate underloaded relays (such as new arrivals), but the underestimate will be corrected within a small number of rounds. Indeed, we found that in our Shadow simulations, a measurement flow would not exceed 4 MB/s even when more spare capacity was available. We also ran a flow-level simulation where each flow was capped at 500 KB/s. This reduced the bandwidth utilization from 14 GB/s to only 1.8 GB/s, while achieving the same estimation accuracy for guards and exits, while increasing the middle estimatio error only slightly, from 22% to 26%.

6 Related Work

Improving the performance of the Tor network has been the subject of much research; we refer the reader to the survey by AlSabah and Goldberg for an overview [1]. Here we summarize related work specifically focusing on relay capacity estimation.

Snader and Borisov proposed using *opportunistic measurements*, where each relay measures the bandwidth of each other relay it communicates with as part of normal operation, and designed EigenSpeed [23], which combines these measurements using principal component analysis to derive a single relay capacity. EigenSpeed was designed to avoid certain types of collusion and misreporting attacks; however, Johnson et al. [18] discovered that it is subject to a number of other attacks that allow colluding adversaries to inflate their bandwidth. They also designed PeerFlow, which is a more robust mechanism to combine opportunistic measurements from relays with provable limits on inflation attacks. These bounds, however, depend on having a fraction of bandwidth being on trusted nodes, and it has slow convergence properties due to its limitations on changing bandwidth values. A point of future research is to investigate whether MLE-style estimation can be used to improve the estimate quality and convergence of opportunistic estimates.

FlashFlow [31] is a new proposal to replace TorFlow. FlashFlow uses several servers that measure a relay simultaneously, generating a large network load intended to max out its capacity. FlashFlow has a guaranteed inflation bound of only 33% but it is based on the assumption that a relay capacity is based on a hard limit that cannot be exceeded, as TorFlow uses traffic that is explicitly labeled for for bandwidth probing. In practice, it is often easier and cheaper to obtain high peak bandwidth capability than sustaining the same bandwidth continuously. For example, in our quick survey of Internet hosting providers, servers with unmetered 10 Gbps traffic cost well over $1,000 per month, whereas a 10 Gbps server with traffic restrictions could be ordered for $200/month or less. FlashFlow also requires coordinating several moderate-bandwidth probe servers, whereas *MLEFlow* can be used with considerably lower capacity probes. (Indeed, in our Shadow simulations we found that a probe to a completely unloaded server maxed out at 4 MB/s, but this did not significantly affect the convergence speed of *MLEFlow*).

Jansen and Johnson evaluate the accuracy of the current Tor capacity estimation algorithms [16] and found that there are significant estimation errors. Similar to our theoretical and simulation results, they find that lower-capacity relays have a larger variance in their relay estimates. Unlike our results, they find that guard relays have lower variation, whereas our simulations show less variance among exit relays. This may be due to patterns of traffic in guard relays that are not captured for the simulator, or the long-term stability of such relays. Their analysis also shows that a too-low observed bandwidth is a large source of error, and can dramatically underestimate the actual bandwidth of large-capacity exit nodes. This suggests that our simulations of $sbws^*$ present an optimistic picture of Tor capacity estimation and in practice the estimation errors will be larger. It also motivates the use of a capacity estimation method that does not use the observed bandwidth.

Several approaches aim to improve load-balancing by detecting and avoiding bottlenecks in real time [2, 5, 11, 32]. These mechanisms still fundamentally rely on relay capacity estimation and their functionality could be improved by using *MLEFlow*. SmarTor [12] aims to decentralize the bandwidth measurement and operation by using trusted execution environment to run the measurements and a smart contract to aggregate them. It does not propose changes to the estimation technique from current TorFlow, but it could be adapted to take advantage of *MLEFlow* instead.

7 Conclusion

We have developed a new method for estimating the relay capacities in the Tor network, *MLEFlow*, based on performing maximum likelihood estimation using a series of bandwidth measurement probes taken over time. Our mathematical analysis showed that *MLEFlow* capacity estimates converge to their true value as the number of users increases, while the estimate variance converges to 0, as the number of observations grows. We showed how to efficiently compute capacity estimates in *MLEFlow* using either closed-form approximations, or quantization; the latter approach allows us to incorporate a model of the measurement noise into the estimates. We validated the performance of *MLEFlow* with extensive simulations using a our custom flow-based simulator and Shadow [15]. Our results show that *MLEFlow* produces much more accurate estimates of relay capacities, which in turn results in much better load balancing of user traffic across the network, as compared with current methods.

8 Acknowledgments

We would like to thank the anonymous referees, as well as our shepherd, Katharina Kohls, for their helpful suggestions. This material is based upon work supported by the National Science Foundation under Grant No. 1739966. This work also benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

References

[1] M. AlSabah and I. Goldberg, "Performance and security improvements for Tor: A survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, p. 32, 2016.

[2] R. Annessi and M. Schmiedecker, "Navigator: Finding faster paths to anonymity," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016, pp. 214–226.

[3] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against Tor," in *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society (WPES)*, 2007, pp. 11–20.

[4] H. Darir, "mleflow," 2021. [Online]. Available: https://github.com/hdarir2/mleflow

[5] H. Darir, H. Sibai, N. Borisov, G. Dullerud, and S. Mitra, "Tightrope: Towards optimal load-balancing of paths in anonymous networks," in *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, 2018, pp. 76–85.

[6] R. Dingledine, "The lifecycle of a new relay," The Tor Project Blog, https://blog.torproject.org/lifecycle-new-relay, Sep. 2013.

[7] ——, "Tor security advisory: "relay early" traffic confirmation attack," https://blog.torproject.org/tor-security-advisory-relay-early-traffic-confirmation-attack, Jul. 2014, tor Blog.

[8] R. Dingledine and N. Mathewson, "Anonymity loves company: Usability and the network effect." in *WEIS*, 2006.

[9] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*. USENIX, 2004, pp. 303–320.

[10] M. Edman and P. Syverson, "As-awareness in tor path selection," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 380–389.

[11] D. Goulet and M. Perry, "Make relays report when they are overloaded," Tor Proposal 328, https://gitlab.torproject.org/tpo/core/torspec/-/blob/master/proposals/328-relay-overload-report.md, Nov. 2020.

[12] A. Greubel, A. Dmitrienko, and S. Kounev, "Smartor: Smarter tor with smart contracts: Improving resilience of topology distribution in the tor network," in *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM, 2018, pp. 677–691. [Online]. Available: https://doi.org/10.1145/3274694.3274722

[13] S. Herbert, S. J. Murdoch, and E. Punskaya, "Optimising node selection probabilities in multi-hop m/d/1 queuing networks to reduce latency of tor," *Electronics letters*, vol. 50, no. 17, pp. 1205–1207, 2014.

[14] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine, "Methodically modeling the tor network," in *5th Workshop on Cyber Security Experimentation and Test (CSET'12)*. Bellevue, WA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/cset12/workshop-program/presentation/Jansen

[15] R. Jansen and N. Hopper, "Shadow: Running Tor in a box for accurate and efficient experimentation," in *Proceedings of the 19th Symposium on Network and Distributed System Security (NDSS)*, 2012.

[16] R. Jansen and A. Johnson, "On the accuracy of Tor bandwidth estimation," in *Passive and Active Measurement Conference (PAM)*, 2021.

[17] R. Jansen, T. Vaidya, and M. Sherr, "Point break: a study of bandwidth denial-of-service attacks against Tor," in *28th USENIX Security Symposium*, 2019, pp. 1823–1840.

[18] A. Johnson, R. Jansen, N. Hopper, A. Segal, and P. Syverson, "PeerFlow: Secure load balancing in Tor," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 74–94, 2017.

[19] juga, "How bandwidth scanners monitor the Tor network," Tor Project Blog, https://blog.torproject.org/aggregation-feed-types/sbws, Apr. 2019.

[20] M. G. Kendall, A. Stuart, and J. K. Ord, *Kendall's Advanced Theory of Statistics*. USA: Oxford University Press, Inc., 1987.

[21] K. Loesing, M. Perry, and A. Gibson, "Bandwidth scanner specification," https://gitweb.torproject.org/torflow.git/tree/NetworkScanners/BwAuthority/README.spec.txt, 2011.

[22] M. Perry, "TorFlow: Tor network analysis," in *Proceedings of the 2nd Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 2009, pp. 1–14.

[23] R. Snader and N. Borisov, "EigenSpeed: Secure peer-to-peer bandwidth evaluation," in *8th International Workshop on Peer-To-Peer Systems*, R. Rodrigues and K. Ross, Eds. Berkeley, CA, USA: USENIX Association, Apr. 2009.

[24] ——, "Improving security and performance in the Tor network through tunable path selection," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 728–741, 2011.

[25] The Tor Project, "Deploying the simple bandwidth scanner," https://sbws.readthedocs.io/en/latest/DEPLOY.html, 2018.

[26] ——, "Differences between Torflow and sbws," https://tpo.pages.torproject.net/network-health/sbws/differences.html, 2020.

[27] ——, "Tor directory protocol, version 3," https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt, 2020.

[28] ——, "Tor metrics: Servers," https://metrics.torproject.org/networksize.html, 2020.

[29] ——, "Tor metrics: Users," https://metrics.torproject.org/userstats-relay-country.html, 2020.

[30] F. Thill, "Hidden service tracking detection and bandwidth cheating in Tor anonymity network," Ph.D. dissertation, University of Luxembourg, 2014.

[31] M. Traudt, R. Jansen, and A. Johnson, "Flashflow: A secure speed test for tor," 2020.

[32] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-aware path selection for Tor," in *International Conference on Financial Cryptography and Data Security*, 2012, pp. 98–113.

[33] P. Winter, R. Ensafi, K. Loesing, and N. Feamster, "Identifying and characterizing sybils in the tor network," in *25th USENIX Security Symposium*, 2016, pp. 1169–1185.

[34] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "The predecessor attack: An analysis of a threat to anonymous communications systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 4, pp. 489–522, 2004.

A Proofs

Deriving equation (4): When evaluating the objective function of the MLE in equation (2), the observation random variable of the j^{th} relay $M_i[j]$ can be written as a function of κ, as if we are assuming $\kappa = C^*[j]$, and the random variable $X_i[j]$ for $i \in [t]$:

$$M_i[j] = \frac{\kappa}{X_i[j]+1}. \tag{18}$$

Recall that we assume that the random variable $X_i[j]$ follows a Poisson distribution with parameter $\lambda_s w_i[j]$ and all users leave at the end of each epoch. Hence, given $w_i[j]$ for $j \in [n]$, the $M_i[j]$'s at different iterations are independent random variables. Thus eq. (3) can be written as the product of the probability of the independent random variables $[M_1[j], ..., M_t[j]]$:

$$C^H_{t+1}[j] = \operatorname*{argmax}_{\kappa\in\mathcal{C}} \Pr_{X_{[t]}[j]\sim \mathrm{Pois}(\lambda_s W_{[t]}[j])}(M_{[t]}[j] = m_{[t]}[j] \,|W_{[t]}[j] = w_{[t]}[j])$$

$$C^H_{t+1}[j] = \operatorname*{argmax}_{\kappa\in\mathcal{C}} \prod_{i=0}^{t} \Pr(M_i[j] = m_i[j] \mid W_i[j] = w_i[j]).$$

Rearranging eq. (18) results in:

$$X_i[j] = \frac{\kappa}{M_i[j]} - 1. \tag{19}$$

When the measurement is made and the observation is fixed, i.e. $M_i[j] = m_i[j]$, the probability in eq. (19) can be expressed in terms of the random variable $X_i[j]$: $X_i[j] = \frac{\kappa}{m_i[j]} - 1$.

$$C^H_{t+1}[j] = \operatorname*{argmax}_{\kappa\in\mathcal{C}} \prod_{i=0}^{t} \Pr(X_i[j] = x_i[j]|\ W_i[j] = w_i[j]) \tag{20}$$

Using the Poisson distribution probability mass function, we can write:

$$\begin{aligned} C^H_{t+1}[j] &= \operatorname*{argmax}_{\kappa\in\mathcal{C}} \prod_{i=0}^{t} e^{-\lambda_s w_i[j]} \frac{1}{(x_i[j])!}(\lambda_s w_i[j])^{x_i[j]} \\ &= \operatorname*{argmax}_{\kappa\in\mathcal{C}} \prod_{i=0}^{t} e^{-\lambda_s w_i[j]} \frac{1}{\left(\frac{\kappa}{m_i[j]}-1\right)!}(\lambda_s w_i[j])^{\frac{\kappa}{m_i[j]}-1} \end{aligned} \tag{21}$$

Theorem 1 (MLE closed form approximation)**.** *For any $j \in [n]$ and $t \in \mathbb{N}$, the MLE estimate of $C^*[j]$ given the weight and observation vectors $w_{[t]}[j]$ and $m_{[t]}[j]$ is*

$$C^H_{t+1}[j] \approx \exp\left(\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}\right), \tag{5}$$

where the approximation tends to equality as the user arrival rate λ_s gets large.

Proof. As we derived in eq. (21), we know that for any $j \in [n]$, the weight at iteration $(t+1)$ should satisfy the following equation:

$$C^H_{t+1}[j] = \operatorname*{argmax}_{\kappa\in\mathcal{C}} \ \prod_{i=0}^{t} e^{-\lambda_s w_i[j]} \frac{1}{\left(\frac{\kappa}{m_i[j]}-1\right)!}(\lambda_s w_i[j])^{\frac{\kappa}{m_i[j]}-1} \tag{22}$$

Since the logarithm function is a strictly increasing function, the maximum likelihood estimate of the capacity of a relay $j \in [n]$ using full history can be found:

$$\begin{aligned} C^H_{t+1}[j] &= \operatorname*{argmax}_{\kappa\in\mathcal{C}} \prod_{i=0}^{t} e^{-\lambda_s w_i[j]} \frac{1}{\left(\frac{\kappa}{m_i[j]}-1\right)!}(\lambda_s w_i[j])^{\frac{\kappa}{m_i[j]}-1} \\ &= \operatorname*{argmax}_{\kappa\in\mathcal{C}} \prod_{i=0}^{t} e^{-\lambda_s w_i[j]} \frac{\frac{\kappa}{m_i[j]}}{\left(\frac{\kappa}{m_i[j]}\right)!}(\lambda_s w_i[j])^{\frac{\kappa}{m_i[j]}}(\lambda_s w_i[j])^{-1} \\ &= \operatorname*{argmax}_{\kappa\in\mathcal{C}} \sum_{i=0}^{t} -\lambda_s w_i[j] + \log\left(\frac{\kappa}{m_i[j]}\right) - \log\left(\left(\frac{\kappa}{m_i[j]}\right)!\right) + \frac{\kappa}{m_i[j]}\log(\lambda_s w_i[j]) - \log(\lambda_s w_i[j]) \end{aligned} \tag{23}$$

Using Stirling's approximation, we have $\log(x!) \approx x\log(x) - x$. Thus substituting in eq. (23):

$$C^H_{t+1}[j] = \operatorname*{argmax}_{\kappa\in\mathcal{C}} \sum_{i=0}^{t} -\lambda_s w_i[j] + \log\left(\frac{\kappa}{m_i[j]}\right) - \frac{\kappa}{m_i[j]}\log\left(\frac{\kappa}{m_i[j]}\right) + \frac{\kappa}{m_i[j]} + \frac{\kappa}{m_i[j]}\log(\lambda_s w_i[j]) - \log(\lambda_s w_i[j]) \tag{24}$$

Hence in order to find $C^H_{t+1}[j]$, we differentiate the right hand side of eq. (24) with respect to κ, and find the value of $C^H_{t+1}[j]$ for which the derivative is zero.

$$\begin{aligned} \sum_{i=0}^{t} \frac{1}{m_i[j]}\frac{1}{\frac{\kappa}{m_i[j]}} - \frac{1}{m_i[j]}\log\left(\frac{\kappa}{m_i[j]}\right) - \frac{1}{m_i[j]} + \frac{1}{m_i[j]} + \frac{1}{m_i[j]}\log(\lambda_s w_i[j]) &= 0 \\ \sum_{i=0}^{t} \frac{1}{\kappa} - \frac{1}{m_i[j]}\log(\kappa) + \frac{1}{m_i[j]}\log(m_i[j]) + \frac{1}{m_i[j]}\log(\lambda_s w_i[j]) &= 0 \\ \sum_{i=0}^{t} \frac{1}{\kappa} - \frac{1}{m_i[j]}\log(\kappa) + \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j]) &= 0 \\ \sum_{i=0}^{t} \frac{1}{m_i[j]}\log(\kappa) = \sum_{i=0}^{t} \frac{1}{\kappa} + \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j]) \\ \log(\kappa)\Big(\sum_{i=0}^{t} \frac{1}{m_i[j]}\Big) = \frac{t+1}{\kappa} + \sum_{i=0}^{t} \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j]) \\ \log(\kappa) = \frac{t+1}{\kappa(\sum_{i=0}^{t} \frac{1}{m_i[j]})} + \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} \\ \log(\kappa) - \frac{t+1}{\kappa(\sum_{i=0}^{t} \frac{1}{m_i[j]})} = \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} \\ \kappa e^{-\frac{t+1}{\kappa(\sum_{i=0}^{t} \frac{1}{m_i[j]})}} = e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\ \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}}{t+1}\kappa e^{-\frac{t+1}{\kappa(\sum_{i=0}^{t} \frac{1}{m_i[j]})}} = \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}}{t+1} e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}\log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \end{aligned} \tag{25}$$

Letting $z = \frac{t+1}{\kappa(\sum_{i=0}^{t} \frac{1}{m_i[j]})}$ in eq. (25), we have:

$$\begin{aligned}
\frac{1}{z}e^{-z} &= \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}}{t+1} e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
\frac{1}{ze^{z}} &= \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]}}{t+1} e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
ze^{z} &= \frac{t+1}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} e^{-\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}
\end{aligned} \tag{26}$$

We know that the inverse image of the function ze^z is the Lambert W function which has real solutions along its principal branch for $z > -\frac{1}{e}$, denoted W_0. Thus we can solve for z:

$$\frac{t+1}{\kappa(\sum_{i=0}^{t} \frac{1}{m_i[j]})} = z = W_0\left(\frac{t+1}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} e^{-\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}\right) \tag{27}$$

And hence solving for κ:

$$C^H_{t+1}[j] = \kappa = \frac{t+1}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} \frac{1}{W_0\left(\frac{t+1}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} e^{-\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}\right)} \tag{28}$$

$$C^H_{t+1}[j] = \frac{t+1}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} \frac{1}{W_0\left(\frac{t+1}{\sum_{i=0}^{t} \frac{1}{m_i[j]}} e^{-\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}\right)}, \tag{29}$$

where W_0 is the Lambert W function along the principal branch. The Lambert W function is the multi-valued complex function $(ze^z)^{-1}$ and W_0 is the unique-valued real function that takes the unique real value of W when $z > \frac{-1}{e}$. Implementations of Lambert function exist in multiple software libraries [7].

W_0 has the following Taylor series expansion for z in the neighborhood of 0: $W_0(z) = z + o(z^2)$. Moreover, the argument of W_0 in Theorem 1 is small if the rate of users arrival to the network λ_s is large enough. Hence, the Taylor expansion around zero is valid and therefore:

$$\begin{aligned}
C^H_{t+1}[j] &\approx e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
&= e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i[j] C^*[j]}{C^*[j]})}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
&= e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i[j]}{C^*[j]}) + \frac{1}{m_i[j]} \log(C^*[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
&= e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i[j]}{C^*[j]})}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} e^{\frac{\log(C^*[j]) \sum_{i=0}^{t} \frac{1}{m_i[j]}}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
&= C^*[j] e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i[j]}{C^*[j]})}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}.
\end{aligned} \tag{30}$$

□

Theorem 2 (Estimates of both methods converge)**.** *For any $j \in [n]$, $t \in \mathbb{N}$, and a method $y \in \{TorFlow\text{-}P, MLEFlow\text{-}CF\}$,*

$$\mathbb{E}[C^y_t[j]] \leq C^*[j]. \tag{9}$$

Moreover, as $t \to \infty$, $\mathbb{E}[C^y_t[j]] \geq C^*[j]\left(1 - \frac{1}{\lambda_s w^*[j]}\right)$. (10)

Proof. **Mean of the estimates when using *MLEFlow-CF*:** In this proof, we consider that weight vectors are generated using method *MF*, i.e. *MLEFlow-CF*. The Lambert W function along the principal branch has the following Taylor series expansion for z in the neighborhood of 0: $W_0(z) = z + o(z)$. Thus for our case:

$$\begin{aligned}
C^{MF}_{t+1}[j] &\approx e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(m_i[j]\lambda_s w_i^{MF}[j])}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}} \\
&= e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{\lambda_s w_i^{MF}[j]}{\frac{1}{m_i[j]}})}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}
\end{aligned} \tag{31}$$

The aim of this proof is to find the expected value of the estimates. We know that $M_i[j] = \frac{C^*[j]}{X_i[j]+1}$ and $\frac{1}{M_i[j]} = \frac{X_i[j]+1}{C^*[j]}$, thus using the fact that the random variable $X_i[j]$ follows a Poisson distribution with parameter $\lambda_s w_i[j]$:

$$E(\frac{1}{M_i[j]}) = E(\frac{X_i[j]+1}{C^*[j]}) = \frac{1}{C^*[j]}(E(X_i[j]) + 1) = \frac{1}{C^*[j]}(\lambda_s w_i[j] + 1) \tag{32}$$

Using first order Taylor expansion approximation around the expected value, we can find that for any

7 https://kite.com/python/docs/mpmath.lambertw

random variable Y, $E[f(Y)] \approx f(E[Y])$ [20]. For a more detailed proof of the derivation of the expected value refer to **proof of theorem 4**. We use this approximation to find the expected value of the estimates $C^{MF}_{t+1}[j]$ by taking $Y = \frac{1}{m_i[j]}$. We also use the fact that given $w^{MF}_i[j]$ for $j \in [n]$, the observations of a relay at different iterations are independent events; thus using eq. (31),

$$\begin{aligned}
E[C^{MF}_{t+1}[j]] &\approx e^{\frac{\sum_{i=0}^{t} \mathbb{E}[\frac{1}{m_i[j]}] \log(\frac{\lambda_s w^{MF}_i[j]}{\mathbb{E}[\frac{1}{m_i[j]}]})}{\sum_{i=0}^{t} \mathbb{E}[\frac{1}{m_i[j]}]}} \\
&= e^{\frac{\sum_{i=0}^{t} \frac{\lambda_s w^{MF}_i[j]+1}{C^*[j]} log(\frac{C^*[j]\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1})}{\sum_{i=0}^{t} \frac{1}{C^*[j]}(\lambda_s w^{MF}_i[j]+1)}} \\
&= e^{\frac{\sum_{i=0}^{t} \frac{1}{C^*[j]}(\lambda_s w^{MF}_i[j]+1) \log(C^*[j](1-\frac{1}{\lambda_s w^{MF}_i[j]+1}))}{\sum_{i=0}^{t} \frac{1}{C^*[j]}(\lambda_s w^{MF}_i[j]+1)}} \\
&= e^{\frac{\sum_{i=0}^{t} (\lambda_s w^{MF}_i[j]+1)(\log(C^*[j])+\log(1-\frac{1}{\lambda_s w^{MF}_i[j]+1}))}{\sum_{i=0}^{t} (\lambda_s w^{MF}_i[j]+1)}} \\
&= C^*[j] e^{\frac{\sum_{i=0}^{t} (\lambda_s w^{MF}_i[j]+1) \log(1-\frac{1}{\lambda_s w^{MF}_i[j]+1})}{\sum_{i=0}^{t} (\lambda_s w^{MF}_i[j]+1)}} \\
&= C^*[j] \prod_{i=0}^{t} (1 - \frac{1}{\lambda_s w^{MF}_i[j]+1})^{\frac{\lambda_s w^{MF}_i[j]+1}{\sum_{\tau=0}^{t} (\lambda_s w^{MF}_\tau[j]+1)}}
\end{aligned} \tag{33}$$

For any $j \in [n]$ and $t \in \mathbb{N}$,

$$\mathbb{E}[C^{MF}_{t+1}[j]] = C^*[j] \prod_{i=0}^{t} (1 - \frac{1}{\lambda_s w^{MF}_i[j]+1})^{\frac{\lambda_s w^{MF}_i[j]+1}{\sum_{\tau=0}^{t} (\lambda_s w^{MF}_\tau[j]+1)}}. \tag{34}$$

Mean of the estimates when using *TorFlow-P*: In this proof we consider that the weight vectors are generated using method T, i.e. *TorFlow-P*. Using the same Taylor expansion used in the proof of eq. (34), we have that $E[f(Y)] \approx f(E[Y])$. We use this approximation to find the expected value of $C^{TF}_{t+1}[j] = m_t[j]\lambda_s w^{TF}_t[j] = \frac{\lambda_s w^{TF}_t[j]}{\frac{1}{m_t[j]}}$ by taking $Y = \frac{1}{m_t[j]}$ and $f(Y) = \frac{\lambda_s w^{TF}_t[j]}{Y}$. Thus using eq. (32),

$$\begin{aligned}
\mathbb{E}[C^{TF}_{t+1}[j]] &= \mathbb{E}[m_t[j]\lambda_s w^{TF}_t[j]] = \mathbb{E}[\frac{\lambda_s w^{TF}_t[j]}{\frac{1}{m_t[j]}}] \\
&\approx (\frac{\lambda_s w^{TF}_t[j]}{\mathbb{E}[\frac{1}{m_t[j]}]}) \\
&= C^*[j] \frac{\lambda_s w^{TF}_t[j]}{\lambda_s w^{TF}_t[j]+1} \\
&= C^*[j](1 - \frac{1}{\lambda_s w^{TF}_t[j]+1})
\end{aligned} \tag{35}$$

For any $j \in [n]$ and $t \in \mathbb{N}$,

$$\mathbb{E}[C^{TF}_{t+1}[j]] = C^*[j](1 - \frac{1}{\lambda_s w^{TF}_t[j]+1}). \tag{36}$$

Proof of Theorem 2 for *MLEFlow-CF*: In the first part of the proof we consider that the weight vectors are generated using method MF, i.e. *MLEFlow-CF*. From eq. (34), we know that,

$$E[C^{MF}_{t+1}[j]] \approx C^*[j] \prod_{i=0}^{t} (1 - \frac{1}{\lambda_s w^{MF}_i[j]+1})^{\frac{\lambda_s w^{MF}_i[j]+1}{\sum_{\tau=0}^{t} (\lambda_s w^{MF}_\tau[j]+1)}} \tag{37}$$

Since $\lambda_s w^{MF}_i[j] > 0$, the multiplicative term in the right hand side of eq. (37) is always between 0 and 1. Let $\delta_i[j] \in [0,1]$ for $i \in [t+1]$ denote the multiplicative term in eq. (37), and hence we can write $\mathbb{E}[C^{MF}_{t+1}[j]] = C^*[j]\delta_{t+1}[j]$. with $\delta_{t+1}[j] \in [0,1]$.

We know that $w^{MF}_i[j] = \frac{C^{MF}_i[j]}{\sum_{k\in[j]} C^{MF}_i[k]} = \frac{C^*[j]\delta_i[j]}{\sum_{k\in[j]} \delta_i[k]C^*[k]}$. Thus,

$$\begin{aligned}
\delta_i[j]w^*[j] &\le \frac{\delta_i[j]}{max_{k\in[n]}\delta_i[k]} w^*[j] \le w^{MF}_i[j] \\
&\le \frac{\delta_i[j]}{\min_{k\in[n]}\delta_i[k]} w^*[j] \le \max_{i\in[t]}(\frac{\delta_i[j]}{\min_{k\in[n]}\delta_i[k]}) w^*[j] = \bar{L}[j]w^*[j]
\end{aligned} \tag{38}$$

The first inequality follows from the fact that $\max_{k\in[n]} \delta_i[k] \le 1$.

In what follows, we should remember that, for $a < 1$ and for $b > 0$, a^b decreases as b increases.

We now have $\delta_i[j]w^*[j] \le w^{MF}_i[j] \le \bar{L}[j]w^*[j]$, replacing in eq. (37), since $(1 - \frac{1}{\lambda_s w^* \delta_i[j]+1}) \le 1$, we can find the following lower bound on $\delta_{t+1}[j]$:

$$1 \ge \delta_{t+1}[j] \ge \prod_{i=0}^{t} (1 - \frac{1}{\lambda_s w^* \delta_i[j]+1})^{\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t} (\lambda_s w^*[j]\delta_\tau+1)}} \tag{39}$$

Assuming that $\delta_i[j]$ is equal to the lower bound for all $i \in [t+1]$ (worst case scenario). We can write $\delta_{t+1}[j]$ as a function of $\delta_t[j]$:

$$\delta_{t+1}[j] = \delta_t[j] \left[\prod_{i=0}^{t-1} ((1 - \frac{1}{\lambda_s w^*\delta_i[j]+1})^{\lambda_s w^*[j]\bar{L}[j]+1})^{\frac{1}{\sum_{\tau=0}^{t} (\lambda_s w^*[j]\delta_\tau[j]+1)} - \frac{1}{\sum_{\tau=0}^{t-1} (\lambda_s w^*[j]\delta_\tau[j]+1)}} \right] (1 - \frac{1}{\lambda_s w^*\delta_t[j]+1})^{\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t} (\lambda_s w^*[j]\delta_\tau[j]+1)}} \tag{40}$$

Since $\lambda_s w^*[j]\delta_i[j] + 1 > 0$ for all $i \in [t]$ then $\frac{1}{\sum_{\tau=0}^{t} (\lambda_s w^*[j]\delta_\tau[j]+1)} < \frac{1}{\sum_{\tau=0}^{t-1} (\lambda_s w^*[j]\delta_\tau[j]+1)}$, thus $\frac{1}{\sum_{\tau=0}^{t} (\lambda_s w^*[j]\delta_\tau[j]+1)} - \frac{1}{\sum_{\tau=0}^{t-1} (\lambda_s w^*[j]\delta_\tau[j]+1)} < 0$. Thus

we can write:

$$\delta_{t+1}[j] = \delta_t[j] \frac{(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}}}{\prod_{i=0}^{t-1}((1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{\lambda_s w^*[j]\bar{L}[j]+1})^{\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}}} \tag{41}$$

We proceed by induction to prove that the sequence $[\delta_0, ..., \delta_{t+1}]$ is a monotonically increasing sequence. Thus assuming $\delta_t > \delta_{t-1} > ... > \delta_0$, we need to show that $\delta_{t+1} > \delta_t$. In order to show that, it is sufficient to show that the term multiplying δ_t in eq. (41) is greater than 1. Since $\delta_t > \delta_{t-1} > ... > \delta_0$ we have for all $i < t$ that

$$1 - \frac{1}{\lambda_s w^*\delta_t[j]+1} > 1 - \frac{1}{\lambda_s w^*\delta_i[j]+1}$$

Since $(\lambda_s w^*[j]\bar{L}[j] + 1)\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)} - \frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big] > 0$ we can write for all $i \in [t-1]$:

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{(\lambda_s w^*[j]\bar{L}[j]+1)\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]} >$$
$$(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{(\lambda_s w^*[j]\bar{L}[j]+1)\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]}$$

By multiplying the above t inequalities (since all terms are positive) we can find that:

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\big[(t)(\lambda_s w^*[j]\bar{L}[j]+1)\big]\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]} >$$
$$\prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{(\lambda_s w^*[j]\bar{L}[j]+1)\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]}$$

Which is equivalent to:

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\Big[\frac{(\lambda_s w^*[j]\bar{L}[j]+1)}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}\frac{(t)(\lambda_s w^*[j]\delta_t[j]+1)}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]} >$$
$$\prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{(\lambda_s w^*[j]\bar{L}[j]+1)\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]}$$

Since we are assuming that $\delta_t > \delta_{t-1} > ... > \delta_0$, we have that $t(\lambda_s w^*[j]\delta_t[j]+1) > \sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)$. Thus,

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\Big[\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]} >$$
$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\Big[\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}\frac{(t)(\lambda_s w^*[j]\delta_t[j]+1)}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]}$$

Thus we now have,

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\Big[\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]} >$$
$$\prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{(\lambda_s w^*[j]\bar{L}[j]+1)\Big[\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}\Big]}$$

Thus we just showed that $\delta_{t+1} > \delta_t$ and the sequence $[\delta_0, ..., \delta_{t+1}]$ is a monotonically increasing sequence.

Since $\delta_i \in [0,1]$ for all $i \in [t+1]$, then $\lambda_s w^*[j]\delta_i[j] \leq \lambda_s w^*[j]$ and thus $1-\frac{1}{\lambda_s w^*[j]\delta_i[j]+1} \leq 1-\frac{1}{\lambda_s w^*[j]+1}$. From which we have

$$\delta_{t+1} = \prod_{i=0}^{t}(1-\frac{1}{\lambda_s w^*[j]\delta_i+1})^{\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau+1)}} \leq$$
$$(1-\frac{1}{\lambda_s w^*[j]+1})^{\frac{(t+1)(\lambda_s w^*[j]\bar{L}[j]+1)}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau+1)}}$$

Since $w^*\delta_i[j] \leq w^*\bar{L}[j]$ for all $i \in [t]$, then,

$$\delta_{t+1} \leq (1-\frac{1}{\lambda_s w^*[j]+1})^{\frac{(t+1)(\lambda_s w^*[j]\bar{L}[j]+1)}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau+1)}} \leq$$
$$(1-\frac{1}{\lambda_s w^*[j]+1})^{\frac{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau+1)}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau+1)}} = (1-\frac{1}{\lambda_s w^*[j]+1})$$

Hence the sequence $[\delta_0, ..., \delta_{t+1}]$ is monotonically increasing and is bounded from above by $(1-\frac{1}{\lambda_s w^*[j]+1})$, hence δ_t converges.

The Cauchy's convergence test states that a series $\delta_t[j]$ converges if and only if for every $\epsilon_1 > 0$, there exist $N \in \mathbb{N}$ such that $|\delta_p[j] - \delta_t[j]| < \epsilon_1$ for all $t, p > N$. Since we showed that $\delta_t[j]$ is an increasing sequence, this is equivalent to saying that for every $\epsilon_2 > 0$, there exist $N \in \mathbb{N}$ such that $1 < \frac{\delta_p[j]}{\delta_t[j]} < 1+\epsilon_2$ for all $p > t > N$. Thus taking $p = t+1$ from eq. (41) we have that,

$$\frac{\delta_{t+1}[j]}{\delta_t[j]} = \frac{(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}}}{\prod_{i=0}^{t-1}((1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{\lambda_s w^*[j]\bar{L}[j]+1})^{\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}}} < 1+\epsilon_2$$

Hence, for every $\epsilon_3 > 0$, there exist $N \in \mathbb{N}$ such that for all $t > N$,

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\frac{\lambda_s w^*[j]\bar{L}[j]+1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}} <$$
$$\prod_{i=0}^{t-1}((1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{\lambda_s w^*[j]\bar{L}[j]+1})^{\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}} + \epsilon_3 \tag{42}$$

For simplicity in the derivation we drop ϵ_3, we will then add it at the end of the derivation. Thus, eq. (42) is equivalent to

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}} < \prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{\frac{1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}}$$

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1})^{\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}} < \prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{(\frac{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}-1)\frac{1}{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}}$$

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1}) < \prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{(\frac{\sum_{\tau=0}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}{\sum_{t-1}^{t}(\lambda_s w^*[j]\delta_\tau[j]+1)}-1)}$$

$$(1-\frac{1}{\lambda_s w^*\delta_t[j]+1}) < \prod_{i=0}^{t-1}(1-\frac{1}{\lambda_s w^*\delta_i[j]+1})^{\frac{\lambda_s w^*[j]\delta_t[j]+1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}}$$

Since we know that $\delta_t > \delta_{t-1} > ... > \delta_0$, thus $(1 - \frac{1}{\lambda_s w^* \delta_i[j]+1})^{\frac{\lambda_s w^*[j]\delta_t[j]+1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}} < (1 - \frac{1}{\lambda_s w^* \delta_i[j]+1})^{\frac{\lambda_s w^*[j]\delta_i[j]+1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}}$ for all $i \in [t-1]$. Thus,

$$(1 - \frac{1}{\lambda_s w^* \delta_t[j]+1}) < \prod_{i=0}^{t-1}(1 - \frac{1}{\lambda_s w^* \delta_i[j]+1})^{\frac{\lambda_s w^*[j]\delta_i[j]+1}{\sum_{\tau=0}^{t-1}(\lambda_s w^*[j]\delta_\tau[j]+1)}}$$

From definition of $\delta_{t+1}[j]$, we can see that the right hand side is equal to $\delta_t[j]$,

$$(1 - \frac{1}{\lambda_s w^* \delta_t[j]+1}) < \delta_t[j]$$

Solving for $\delta_t[j]$, We can hence say that for every $\epsilon_4 > 0$, there exist $N \in \mathbb{N}$ such that for all $t > N$,

$$\delta_t[j] + \epsilon_4 > 1 - \frac{1}{\lambda_s w^*}$$

Thus as $t \to \infty$ we have,

$$1 \geq \delta_t[j] \geq 1 - \frac{1}{\lambda_s w^*} \tag{43}$$

Taking $\epsilon = \frac{1}{\lambda_s w^*[j]}$, prove the theorem.

Proof of Theorem 2 for *TorFlow-P*: Similarly for *TorFlow-P*, we have from eq. (36) that the expected value of the estimates of the capacities using *TorFlow-P* is,

$$\mathbb{E}[C_{t+1}^T[j]] = C^*[j](1 - \frac{1}{\lambda_s w_t^T[j] + 1}) \tag{44}$$

In this part of the proof, we consider that weight vectors are generated using method T. Since $\lambda_s w_t^{TF}[j] > 0$ for all $t \in \mathbb{N}$, the multiplicative term in the right hand side of eq. (44) is always between 0 and 1. Let $\delta'_{t+1}[j] \in [0,1]$ for $t \in \mathbb{N}$ denote the multiplicative term in eq. (44), and hence we can write $\mathbb{E}[C_{t+1}^{TF}[j]] = C^*[j]\delta'_{t+1}[j]$. with $\delta'_{t+1}[j] \in [0,1]$.

We know that $w_t^{TF}[j] = \frac{C_t^{TF}[j]}{\sum_{k\in[j]} C_t^{TF}[k]} = \frac{C^*[j]\delta'_t[j]}{\sum_{k\in[j]} \delta'_t[k]C^*[k]}$. Thus,

$$\delta'_t[j]w^*[j] \leq \frac{\delta'_t[j]}{max_{k\in[n]}\delta'_t[k]} w^*[j] \leq w_t^{TF}[j] \tag{45}$$

The first inequality follows from the fact that $max_{k\in[n]}\delta'_t[k] \leq 1$.

Thus we can find the following lower bound on $\delta'_{t+1}[j]$:

$$1 \geq \delta'_{t+1}[j] \geq 1 - \frac{1}{\lambda_s w^* \delta'_t[j] + 1} \tag{46}$$

Assuming that $\delta'_{t+1}[j]$ is equal to the lower bound for all $t \in \mathbb{N}$ (worst case scenario). We can find the ratio of $\delta'_{t+1}[j]$ over $\delta'_t[j]$,

$$\frac{\delta'_{t+1}[j]}{\delta'_t[j]} = \frac{1 - \frac{1}{\lambda_s w^* \delta'_t[j]+1}}{1 - \frac{1}{\lambda_s w^* \delta'_{t-1}[j]+1}} \tag{47}$$

We proceed by induction to prove that the sequence $[\delta'_0, ..., \delta'_{t+1}]$ is a monotonically increasing sequence. Thus assuming $\delta'_t > \delta'_{t-1} > ... > \delta'_0$, we need to show that $\delta'_{t+1} > \delta'_t$. In order to show that, it is sufficient to show that the ratio in eq. (47) is greater than or equal to 1. Since $\delta'_t > \delta'_{t-1}$, we have for all $t \in \mathbb{N}$ that $1 - \frac{1}{\lambda_s w^* \delta'_t[j]+1} > 1 - \frac{1}{\lambda_s w^* \delta'_{t-1}[j]+1}$, and thus $\frac{\delta'_{t+1}[j]}{\delta'_t[j]} > 1$. We hence showed that the sequence $[\delta'_0, ..., \delta'_{t+1}]$ is a monotonically increasing sequence.

Since $\delta'_t \in [0,1]$ for all $t \in \mathbb{N}$, then $\lambda_s w^*[j]\delta'_t[j] \leq \lambda_s w^*[j]$ and thus $1 - \frac{1}{\lambda_s w^*[j]\delta'_t[j]+1} \leq 1 - \frac{1}{\lambda_s w^*[j]+1}$. From which we have that $\delta_{t+1}[j] \leq 1 - \frac{1}{\lambda_s w^*[j]+1}$ for all $t \in \mathbb{N}$. Hence the sequence $[\delta'_0, ..., \delta'_{t+1}]$ is monotonically increasing and is bounded from above by $(1 - \frac{1}{\lambda_s w^*[j]+1})$, hence δ'_t converges.

The Cauchy's convergence test states that a series $\delta'_t[j]$ converges if and only if for every $\epsilon'_1 > 0$, there exist $N \in \mathbb{N}$ such that $|\delta'_p[j] - \delta'_t[j]| < \epsilon'_1$ for all $t, p > N$. Since we showed that $\delta'_t[j]$ is an increasing sequence, this is equivalent to saying that for every $\epsilon'_2 > 0$, there exist $N \in \mathbb{N}$ such that $1 < \frac{\delta'_p[j]}{\delta'_t[j]} < 1 + \epsilon_2$ for all $p > t > N$. Thus taking $p = t + 1$ from eq. (47) we have that,

$$\frac{\delta'_{t+1}[j]}{\delta'_t[j]} = \frac{1 - \frac{1}{\lambda_s w^* \delta'_t[j]+1}}{1 - \frac{1}{\lambda_s w^* \delta'_{t-1}[j]+1}} < 1 + \epsilon'_2$$

Hence, for every $\epsilon'_3 > 0$, there exist $N \in \mathbb{N}$ such that for all $t > N$,

$$1 - \frac{1}{\lambda_s w^* \delta'_t[j] + 1} < 1 - \frac{1}{\lambda_s w^* \delta'_{t-1}[j] + 1} + \epsilon'_3 = \delta'_t + \epsilon'_3. \tag{48}$$

Solving for $\delta'_t[j]$, We can hence say that for every $\epsilon'_4 > 0$, there exist $N \in \mathbb{N}$ such that for all $t > N$,

$$\delta'_t[j] + \epsilon'_4 > 1 - \frac{1}{\lambda_s w^*}$$

Thus as $t \to \infty$ we have,

$$1 \geq \delta'_t[j] \geq 1 - \frac{1}{\lambda_s w^*} \tag{49}$$

Taking $\epsilon = \frac{1}{\lambda_s w^*[j]}$, prove the theorem. □

Theorem 4 (Variance of *TorFlow-P* > Variance of *MLEFlow-CF*)**.** *For λ_s big enough, for all $j \in [n]$, and starting with the same initial weights, $Var[C_1^{MF}[j]] = Var[C_1^{TF}[j]]$ and for $t > 1$,*

$$\frac{Var[C_{t+1}^{TF}[j]]}{Var[C_{t+1}^{MF}[j]]} > \frac{t+1}{\zeta}. \tag{11}$$

with $\zeta = 1 + \frac{1}{e^2} + \frac{2}{e}$.

Moreover, as $t \to \infty$, $Var[C_t^{MF}[j]] \to 0$.

Proof. **Finding an upper bound for the variance of the estimates when using *MLEFlow-CF*:**

In this proof, we consider that weight vectors are generated using method *MF*. First, from eq. (32) we have,

$$E(\frac{1}{M_i[j]}) = E(\frac{X_i[j]+1}{C^*[j]}) = \frac{1}{C^*[j]}(E(X_i[j])+1) = \frac{1}{C^*[j]}(\lambda_s w_i[j]+1) \tag{50}$$

Analogously, we can find,

$$Var(\frac{1}{M_i[j]}) = Var(\frac{X_i[j]+1}{C^*[j]}) = \frac{1}{C^*[j]^2}Var[X_i[j]+1] = \frac{\lambda_s w_i[j]}{C^*[j]^2} \tag{51}$$

We have from theorem 1 the closed form of our estimate at time t for any $j \in [n]$.

$$C_{t+1}^{MF}[j] \approx C^*[j] e^{\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})}{\sum_{i=0}^{t} \frac{1}{m_i[j]}}}.$$

Using a first order Taylor expansion approximation [20], we can find that for any random variable Y, $E[f(Y)] \approx f(E[Y])$ and $Var[f(Y)] \approx (f'(E[Y]))^2 Var[Y]$. We use this approximation to find the expected value and variance of $\frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j])}{C^*[j]}$ by taking $Y = \frac{1}{m_i[j]}$ and $f(Y) = -Y log(\frac{C^*[j]}{\lambda_s w_i^{MF}[j]} Y)$. Thus using eq. (50) and eq. (51),

$$\begin{aligned} Var[\frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})] &= (\log(\frac{\lambda_s w_i^{MF}[j]}{\lambda_s w_i^{MF}[j]+1}) - 1)^2 \frac{\lambda_s w_i^{MF}[j]}{C^*[j]^2} \\ E[\frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})] &= \frac{\lambda_s w_i^{MF}[j]+1}{C^*[j]} \log(\frac{\lambda_s w_i^{MF}[j]}{\lambda_s w_i^{MF}[j]+1}) \end{aligned} \tag{52}$$

Given $w_i^{MF}[j]$ for $j \in [n]$, the observations of a relay at different iterations are independent events. Thus we can find, with

$$\begin{aligned} (\sigma_t^1[j])^2 &= Var[\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})] = \sum_{i=0}^{t} (\log(\frac{\lambda_s w_i^{MF}[j]}{\lambda_s w_i^{MF}[j]+1}) - 1)^2 \frac{\lambda_s w_i^{MF}[j]}{C^*[j]^2} \\ \mu_t^1[j] &= E[\sum_{i}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})] = \sum_{i=0}^{t} \frac{\lambda_s w_i^{MF}[j]+1}{C^*[j]} \log(\frac{\lambda_s w_i^{MF}[j]}{\lambda_s w_i^{MF}[j]+1}) \end{aligned} \tag{53}$$

From eq. (50) and eq. (51), we also have,

$$\begin{aligned} (\sigma_t^2[j])^2 &= Var[\sum_{i}^{t} \frac{1}{m_i[j]}] = \sum_{i=0}^{t} \frac{\lambda_s w_i^{MF}[j]}{C^*[j]^2} \\ \mu_t^2[j] &= E[\sum_{i}^{t} \frac{1}{m_i[j]}] = \sum_{i=0}^{t} \frac{1}{C^*[j]}(\lambda_s w_i^{MF}[j]+1) \end{aligned} \tag{54}$$

Using the same approach applied earlier we can find that for any two random variables Y and Z we have, $E[g(Y,Z)] \approx g(E(Y),E(Z))$ and $Var[g(Y,Z)] \approx g'_y(E[Y],E[Z])^2 Var[Y] + 2g'_y(E[Y],E[Z])g'_z(E[Y],E[Z])Cov(Y,Z) + g'_z(E[Y],E[Z])^2 Var[Z]$. Using Cauchy-Schwarz inequality we can upper bound the $|Cov(Y,Z)|$ by $\sqrt{Var[Y]\,Var[Z]}$. Hence, we have,

$$\begin{aligned} Var[g(Y,Z)] \le\ & g'_y(E[Y],E[Z])^2 Var[Y] + 2g'_y(E[Y],E[Z])g'_z(E[Y],E[Z])\sqrt{Var[Y]\,Var[Z]} \\ & + g'_z(E[Y],E[Z])^2 Var[Z] \end{aligned} \tag{55}$$

For the special case of $g(Y,Z) = \frac{Y}{Z}$, we have,

$$\begin{aligned} Var[\frac{Y}{Z}] &\le \frac{E[Y]^2}{E[Z]^2}(\frac{Var[Y]}{E[Y]^2} - 2\frac{\sqrt{Var[Y]\,Var[Z]}}{E[Y]E[Z]} + \frac{Var[Z]}{E[Z]^2}) \\ &= \frac{E[Y]^2}{E[Z]^2}(\frac{\sigma[Y]}{E[Y]} - \frac{\sigma[Z]}{E[Z]})^2 \end{aligned} \tag{56}$$

Taking $Y = \sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})$ and $Z = \sum_{i}^{t} \frac{1}{m_i[j]}$, we can find that,

$$\begin{aligned} Var[\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})}{\sum_{i}^{t} \frac{1}{m_i[j]}}] &\le \frac{(\mu_t^1[j])^2}{(\mu_t^2[j])^2}(\frac{\sigma_t^1[j]}{\mu_t^1[j]} - \frac{\sigma_t^2[j]}{\mu_t^2[j]})^2 \\ &= V[j] \end{aligned} \tag{57}$$

Similarly as before,

$$\mathbb{E}[\frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})}{\sum_{i}^{t} \frac{1}{m_i[j]}}] \approx \frac{\mu_t^1[j]}{\mu_t^2[j]} \tag{58}$$

Using similar argument and taking $X = \frac{\sum_{i=0}^{t} \frac{1}{m_i[j]} \log(\frac{m_i[j]\lambda_s w_i^{MF}[j]}{C^*[j]})}{\sum_{i}^{t} \frac{1}{m_i[j]}}$ and $f(X) = C^*[j]e^X$, we can find $E[C^*[j]e^X] \approx C^*[j]e^{E[X]}$ and $Var[C^*[j]e^X] \approx (C^*[j]e^{E[X]})^2 Var[X]$. Thus,

$$Var[C_{t+1}[j]] \le C^*[j]^2 (e^{\frac{\mu_t^1[j]}{\mu_t^2[j]}})^2 V[j] \tag{59}$$

$$E[C_{t+1}[j]] \approx C^*[j] e^{\frac{\mu_t^1[j]}{\mu_t^2[j]}} \tag{60}$$

For any $j \in [n]$ and $t \in \mathbb{N}$, the variance of $C^{MF}_{t+1}[j]$ is upper bounded by,

$$Var[C^{MF}_{t+1}[j]] \leq C^*[j]^2 (e^{\frac{\mu^1_t[j]}{\mu^2_t[j]}})^2 V[j], \tag{61}$$

where

$$V[j] = \frac{(\mu^1_t[j])^2}{(\mu^2_t[j])^2}(\frac{\sigma^1_t[j]}{\mu^1_t[j]} - \frac{\sigma^2_t[j]}{\mu^2_t[j]})^2,$$

$$(\sigma^1_t[j])^2 = \sum_{i=0}^{t}(\log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) - 1)^2 \frac{\lambda_s w^{MF}_i[j]}{C^*[j]^2},$$

$$\mu^1_t[j] = \sum_{i=0}^{t} \frac{\lambda_s w^{MF}_i[j]+1}{C^*[j]} \log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}),$$

$$(\sigma^2_t[j])^2 = \sum_{i=0}^{t} \frac{\lambda_s w^{MF}_i[j]}{C^*[j]^2}, \text{ and}$$

$$\mu^2_t[j] = \sum_{i=0}^{t} \frac{1}{C^*[j]}(\lambda_s w^{MF}_i[j]+1).$$

Finding the variance of the estimates when using *TorFlow-P*: In this proof we consider that the weight vectors are generated using method *TF*, i.e. *TorFlow-P*. Using the same Taylor expansion used in the proof of eq. (61), we have that $E[f(Y)] \approx f(E[Y])$ and $Var[f(Y)] \approx (f'(E[Y])^2\, Var[Y]$. We use this approximation to find the variance of $C^{TF}_{t+1}[j] = m_t[j]\lambda_s w^{TF}_t[j] = \frac{\lambda_s w^{TF}_t[j]}{\frac{1}{m_t[j]}}$ by taking $Y = \frac{1}{m_t[j]}$ and $f(Y) = \frac{\lambda_s w^{TF}_t[j]}{Y}$. Thus using eq. (50) and eq. (51),

$$\begin{aligned} Var[m_t[j]\lambda_s w^{TF}_t[j]] &= Var[\frac{\lambda_s w^{TF}_t[j]}{\frac{1}{m_t[j]}}] \\ &\approx (\frac{\lambda_s w^{TF}_t[j]}{\mathbb{E}[\frac{1}{m_t[j]}]})^2\, Var[\frac{1}{m_t[j]}] \\ &= (\frac{C^*[j]^2 \lambda_s w^{TF}_t[j]}{(\lambda_s w^{TF}_t[j]+1)^2})^2 \frac{\lambda_s w^{TF}_t[j]}{C^*[j]^2} \\ &= C^*[j]^2 \frac{(\lambda_s w^{TF}_t[j])^3}{(\lambda_s w^{TF}_t[j]+1)^4} \end{aligned} \tag{62}$$

For any $j \in [n]$ and $t \in \mathbb{N}$, the variance of $C^{TF}_{t+1}[j]$ is as follows:

$$Var[C^{TF}_{t+1}[j]] = C^*[j]^2 \frac{(\lambda_s w^{TF}_t[j])^3}{(\lambda_s w^{TF}_t[j]+1)^4}. \tag{63}$$

Comparing the two variances: From eq. (61), we have the upper bound on the variance of the estimates of *MLEFlow-CF*,

$$Var[C^{MF}_{t+1}[j]] \leq C^*[j]^2 (e^{\frac{\mu^1_t[j]}{\mu^2_t[j]}})^2 V[j], \tag{64}$$

Since $\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1} < 1$, $\log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) < 0$ for all $i \in [t]$ and all $j \in [n]$, thus,

$$\mu^1_t[j] = \sum_{i=0}^{t} \frac{\lambda_s w^{MF}_i[j]+1}{C^*[j]} log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) < 0 \tag{65}$$

Since $\mu^2_t[j] = \sum_{i=0}^{t} \frac{1}{C^*[j]}(\lambda_s w^{MF}_i[j]+1) > 0$, then

$$e^{\frac{\mu^1_t[j]}{\mu^2_t[j]}} < 1. \tag{66}$$

Since $\sigma^1_t[j]$ and $\sigma^2_t[j]$ are the standard deviation of two random variables, $\sigma^1_t[j] > 0$ and $\sigma^2_t[j] > 0$ for all $j \in [n]$. From eq. (61),

$$\begin{aligned} V[j] &= \frac{(\mu^1_t[j])^2}{(\mu^2_t[j])^2}(\frac{\sigma^1_t[j]}{\mu^1_t[j]} - \frac{\sigma^2_t[j]}{\mu^2_t[j]})^2 \\ &= \frac{(\sigma^1_t[j])^2}{(\mu^2_t[j])^2} + \frac{(\mu^1_t[j])^2(\sigma^2_t[j])^2}{(\mu^2_t[j])^4} - 2\frac{\mu^1_t[j]\sigma^1_t[j]\sigma^2_t[j]}{(\mu^2_t[j])^3} \end{aligned} \tag{67}$$

Since for any $y > 0$, $\log(y) \geq 1 - \frac{1}{y}$, we have that $0 > \log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) - 1 \geq -\frac{\lambda_s w^{MF}_i[j]+1}{\lambda_s w^{MF}_i[j]}$, hence $(\log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) - 1)^2 \leq \frac{(\lambda_s w^{MF}_i[j]+1)^2}{(\lambda_s w^{MF}_i[j])^2}$ for all $i \in [t]$ and $j \in [n]$. Thus from eq. (61),

$$\begin{aligned} (\sigma^1_t[j])^2 &= \sum_{i=0}^{t}(\log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) - 1)^2 \frac{\lambda_s w^{MF}_i[j]}{C^*[j]^2} \\ &\leq \frac{1}{C^*[j]^2} \sum_{i=0}^{t} \frac{(\lambda_s w^{MF}_i[j]+1)^2}{\lambda_s w^{MF}_i[j]} \end{aligned} \tag{68}$$

Hence taking the square root we have,

$$\sigma^1_t[j] \leq \frac{1}{C^*[j]} \sqrt{\sum_{i=0}^{t} \frac{(\lambda_s w^{MF}_i[j]+1)^2}{\lambda_s w^{MF}_i[j]}} \tag{69}$$

Using simple mathematical arguments (table of variation of the difference), we can find that for $y > 0$, $\log(y) > \frac{-1/e}{y}$. Thus, $\log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) \geq -\frac{1}{e}\frac{\lambda_s w^{MF}_i[j]+1}{\lambda_s w^{MF}_i[j]}$. From eq. (61),

$$\begin{aligned} 0 > \mu^1_t[j] &= \sum_{i=0}^{t} \frac{\lambda_s w^{MF}_i[j]+1}{C^*[j]} \log(\frac{\lambda_s w^{MF}_i[j]}{\lambda_s w^{MF}_i[j]+1}) \\ &> -\frac{1/e}{C^*[j]} \sum_{i=0}^{t} \frac{(\lambda_s w^{MF}_i[j]+1)^2}{\lambda_s w^{MF}_i[j]} \end{aligned} \tag{70}$$

Thus,

$$-\mu^1_t[j] < \frac{1/e}{C^*[j]} \sum_{i=0}^{t} \frac{(\lambda_s w^{MF}_i[j]+1)^2}{\lambda_s w^{MF}_i[j]} \tag{71}$$

And,

$$(\mu_t^1[j])^2 < \frac{1/e^2}{C^*[j]^2}(\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]})^2 \tag{72}$$

From eq. (61), we have,

$$(\sigma_t^2[j])^2 = Var[\sum_i^t \frac{1}{m_i[j]}] = \sum_{i=0}^{t}\frac{\lambda_s w_i^{MF}[j]}{C^*[j]^2} \tag{73}$$

$$\mu_t^2[j] = \mathbb{E}[\sum_i^t \frac{1}{m_i[j]}] = \sum_{i=0}^{t}\frac{1}{C^*[j]}(\lambda_s w_i^{MF}[j]+1) \tag{74}$$

Taking the square root of the equation of $(\sigma_t^2[j])^2$ above, we can find that,

$$\sigma_t^2[j] = \frac{1}{C^*[j]}\sqrt{\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]} \tag{75}$$

Using eq. (68) and eq. (74), we can find,

$$\frac{(\sigma_t^1[j])^2}{(\mu_t^2[j])^2} \leq \frac{\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]}}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^2} \tag{76}$$

Using eq. (72), eq. (73) and eq. (74), we can find,

$$\begin{aligned}\frac{(\mu_t^1[j])^2(\sigma_t^2[j])^2}{(\mu_t^2[j])^4} &< \frac{1}{e^2}\frac{(\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]})^2(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^4} \\ &= \frac{1}{e^2}\frac{(\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]})^2}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3}\end{aligned} \tag{77}$$

Using eq. (71), eq. (75), eq. (69) and eq. (74), we can find that,

$$\begin{aligned}-2\frac{\mu_t^1[j]\sigma_t^1[j]\sigma_t^2[j]}{(\mu_t^2[j])^3} &= 2\frac{(-\mu_t^1[j])\sigma_t^1[j]\sigma_t^2[j]}{(\mu_t^2[j])^3} \\ &< \frac{2}{e}\frac{(\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]})(\sqrt{\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]}})(\sqrt{\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]})}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3}\end{aligned} \tag{78}$$

Using eq. (76), eq. (77) and eq. (78), we can hence find an upper bound on $V[j]$,

$$\begin{aligned}V[j] < &\frac{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]}}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3} \\ &+\frac{1}{e^2}\frac{(\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]})^2}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3} \\ &+\frac{2}{e}\frac{(\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]})(\sqrt{\sum_{i=0}^{t}\frac{(\lambda_s w_i^{MF}[j]+1)^2}{\lambda_s w_i^{MF}[j]}})(\sqrt{\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]})}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3}\end{aligned} \tag{79}$$

Factorising the numerator by λ_s,

$$\begin{aligned}V[j] < &\frac{(\lambda_s)^2(\sum_{i=0}^{t}w_i^{MF}[j]+\frac{1}{\lambda_s})\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3} \\ &+\frac{1}{e^2}\frac{(\lambda_s)^2(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})^2}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3} \\ &+\frac{2}{e}\frac{(\lambda_s)^2(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})(\sqrt{\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}})(\sqrt{\sum_{i=0}^{t}w_i^{MF}[j]})}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3}\end{aligned} \tag{80}$$

From from eq. (80), eq. (66) and eq. (61), we replace $V[j]$ and $e^{\frac{\mu_t^1[j]}{\mu_t^2[j]}}$ by their upper bounds to get

$$\begin{aligned}Var[C_{t+1}^{MF}[j]] < C^*[j]^2(&\frac{(\lambda_s)^2(\sum_{i=0}^{t}w_i^{MF}[j]+\frac{1}{\lambda_s})\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3} \\ &+\frac{1}{e^2}\frac{(\lambda_s)^2(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})^2}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3} \\ &+\frac{2}{e}\frac{(\lambda_s)^2(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})(\sqrt{\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}})(\sqrt{\sum_{i=0}^{t}w_i^{MF}[j]})}{(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3}\end{aligned} \tag{81}$$

We know that $Var[C_{t+1}^{TF}[j]] = C^*[j]^2\frac{(\lambda_s w_t^{TF}[j])^3}{(\lambda_s w_t^{TF}[j]+1)^4}$, thus we now have,

$$\begin{aligned}\frac{Var[C_{t+1}^{TF}[j]]}{Var[C_{t+1}^{MF}[j]]} > &\frac{(\lambda_s w_t^{TF}[j])^3(\sum_{i=0}^{t}\lambda_s w_i^{MF}[j]+1)^3}{(\lambda_s w_t^{TF}[j]+1)^4(\lambda_s)^2((\sum_{i=0}^{t}w_i^{MF}[j]+\frac{1}{\lambda_s})(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}) \\ &+\frac{1}{e^2}(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})^2 \\ &+\frac{2}{e}(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})(\sqrt{\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}})(\sqrt{\sum_{i=0}^{t}w_i^{MF}[j]}))\end{aligned} \tag{82}$$

Which is equivalent to,

$$\begin{aligned}\frac{Var[C_{t+1}^{TF}[j]]}{Var[C_{t+1}^{MF}[j]]} > &\frac{(\lambda_s)^6(w_t^{TF}[j])^3(\sum_{i=0}^{t}w_i^{MF}[j]+\frac{1}{\lambda_s})^3}{(\lambda_s)^6(w_t^{TF}[j]+\frac{1}{\lambda_s})^4((\sum_{i=0}^{t}w_i^{MF}[j]+\frac{1}{\lambda_s})(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}) \\ &+\frac{1}{e^2}(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})^2 \\ &+\frac{2}{e}(\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]})(\sqrt{\sum_{i=0}^{t}\frac{(w_i^{MF}[j]+\frac{1}{\lambda_s})^2}{w_i^{MF}[j]}})(\sqrt{\sum_{i=0}^{t}w_i^{MF}[j]}))\end{aligned} \tag{83}$$

Thus for λ_s big enough we have,

$$\frac{Var[C_{t+1}^{TF}[j]]}{Var[C_{t+1}^{MF}[j]]} > \frac{(w_t^{TF}[j])^3(\sum_{i=0}^{t}w_i^{MF}[j])^3}{(w_t^{TF}[j])^4((\sum_{i=0}^{t}w_i^{MF}[j])^2+\frac{1}{e^2}(\sum_{i=0}^{t}w_i^{MF}[j])^2+\frac{2}{e}(\sum_{i=0}^{t}w_i^{MF}[j])^2)} \tag{84}$$

From eq. (34) and eq. (36), for λ_s big enough, we have $E[C_i^{MF}[j]] \sim C^*[j]$ and $E[C_t^{TF}[j]] \sim C^*[j]$. And thus for λ_s big enough, we have $E[w_i^{MF}[j]] \sim w^*[j]$ and $E[w_t^{TF}[j]] \sim w^*[j]$. Hence from eq. (84),

$$\begin{aligned}\frac{Var[C_{t+1}^{TF}[j]]}{Var[C_{t+1}^{MF}[j]]} &> \frac{(w^*[j])^6(t+1)^3}{(1+\frac{1}{e^2}+\frac{2}{e})(t+1)^2(w^*[j])^6} \\ &= \frac{t+1}{(1+\frac{1}{e^2}+\frac{2}{e})}\end{aligned} \tag{85}$$

Thus for $t \geq 1$, we have that $\frac{Var[C^{TF}_{t+1}[j]]}{Var[C^{MF}_{t+1}[j]]} > 1$, and thus the variance of *TorFlow-P* is greater than the variance of *MLEFlow-CF*. We also know that at $t = 0$ and starting at the same initial weight vector, both algorithms are equivalent since they both are a one step maximum likelihood optimization. Thus for λ_s big enough, for all $t \geq 1$, the variance of the ML estimation is at most equal to the variance of *TorFlow-P*.

Proof (Convergence of variance of *MLEFlow-CF*) As $t \to \infty$, $Var[C^{MF}_t[j]] \to 0$.

$Var[C^{TF}_t[j]]$ is bounded since λ_s is constant and $w_t \in [0, 1]$. Substituting these in equation (63), results in $Var[C^{TF}_1[j]] \leq C^*[j]^2\lambda_s^3$. Substituting this in equation (11), results in the statement. □

Deriving equation (15): Since $[M_0[j], ..., M_t[j]]$ are independent random variables, we can write the estimate as:

$$C^N_{t+1}[j] = \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{t} f_o(M_i[j] = m_i[j] \mid w_i[j]) \quad (86)$$

From eq. (12), given $C[j] = \kappa$ and $w_i[j]$, we let $h_C(X_i[j], Y_i[j]) = M_i[j] = \frac{\kappa Y_i[j]}{X_i[j]+1}$ for $i \in [t]$. Thus from eq. (86), replacing $M_i[j]$ by $h_C(X_i[j], Y_i[j])$:

$$C^N_{t+1}[j] = \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{t} f_o(h_C(X_i[j], Y_i[j]) = m_i[j]) \quad (87)$$

We know from the law of total probability that for $i \in [t]$:

$$f_o(h_C(X_i[j], Y_i[j]) = m_i[j]) = \int_{y \in Y_i[j]} f_o(h_C(X_i[j], Y_i[j]) = m_i[j] \mid Y_i[j] = y) f_{normal}(Y_i[j] = y) \quad (88)$$

where f_{normal} refers to the probability density function of the normal distribution with mean 1 and standard deviation σ_e.

Replacing eq. (88) in eq. (87), we get:

$$C^N_{t+1}[j] = \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{t} \int_{y \in Y_i[j]} f_o(h_C(X_i[j], Y_i[j]) = m_i[j] \mid Y_i[j] = y) f_{normal}(Y_i[j] = y) \quad (89)$$

From the definition of $h_C(X_i[j], Y_i[j])$, given that $Y_i[j] = y$ and $h_C(X_t[j], Y_t[j]) = m_t[j]$, we can find that:

$$f_o(h_C(X_t[j], Y_t[j]) = m_t[j] \mid Y_t[j] = y) = \begin{cases} e^{-\lambda_s w_i[j]} \frac{1}{(x)!} (\lambda_s w_i[j])^x \delta(y) \\ \quad where \;\; x \in [0, ..., n] \\ \quad and \;\; y = \frac{m_i[j](x+1)}{\kappa} \in [y_{min}, y_{max}] \end{cases} \quad (90)$$

Thus using eq. (90) in eq. (89), we can formulate the estimate to be:

$$\begin{aligned} C^N_{t+1}[j] &= \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{t} \sum_{x=0}^{n} e^{-\lambda_s w_i[j]} \frac{1}{(x)!} (\lambda_s w_i[j])^x f_{normal}(Y_i[j] = y) \\ &= \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{t} \sum_{x=0}^{n} e^{-\lambda_s w_i[j]} \frac{1}{(x)!} (\lambda_s w_i[j])^x \frac{1}{\sigma_e \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\frac{m_i[j](x+1)}{\kappa} - 1}{\sigma_e}\right)^2} \end{aligned} \quad (91)$$

If we consider the expression of x in terms of y, $x = \frac{\kappa y}{m_i[j]} - 1$, we see that it is a continuous and increasing function of y. Since $y \in [y_{min}, y_{max}]$, thus we can find the maximum value of x for $y = y_{max}$ and the minimum value for $y = y_{min}$. Thus given $m_t[j]$, we can find $x_{max,i}[j] = \lfloor \frac{\kappa y_{max}}{m_i[j]} - 1 \rfloor$ and $x_{min,i}[j] = \lceil \frac{\kappa y_{min}}{m_i[j]} - 1 \rceil$ for all $i \in [t]$. Thus in order to find the maximum likelihood estimate:

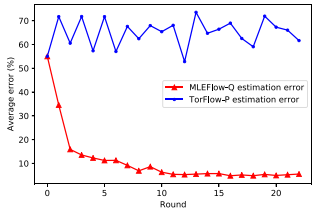
$$C^N_{t+1}[j] = \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{t} \sum_{x=x_{min,i}[j]}^{x_{max,i}[j]} e^{-\lambda_s w_i[j]} \frac{1}{(x)!} (\lambda_s w_i[j])^x \frac{1}{\sigma_e \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\frac{m_i[j](x+1)}{\kappa} - 1}{\sigma_e}\right)^2} \quad (92)$$

Applying the same strategy using the logarithm function,and since $\frac{1}{\sigma_e \sqrt{2\pi}}$ is a positive constant, the maximum likelihood estimate of C can be found:

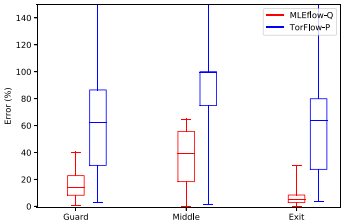
$$C^N_{t+1}[j] = \underset{\kappa \in \mathcal{C}}{\operatorname{argmax}} \sum_{i=0}^{t} \log \left[\sum_{x=x_{min,i}[j]}^{x_{max,i}[j]} e^{-\lambda_s w_i[j]} \frac{1}{(x)!} (\lambda_s w_i[j])^x e^{-\frac{1}{2}\left(\frac{\frac{m_i[j](x+1)}{\kappa} - 1}{\sigma_e}\right)^2} \right] \quad (93)$$

B Unbiased Python Simulations

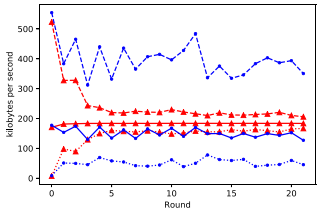
In order to show that the Python implementation of *MLEFlow* is not biased, we simulated the same 3% network using both our Python simulator and Shadow. The 3% network contains 196 Tor relays (61 guards, 109 middle relays, and 26 exits). The total throughput is 1.3 GB/s (guards: 713 MB/s, middles: 252 MB/s, exits: 359 MB/s). We simulated Algorithm 1 in Python using $\lambda_s = 2000$ and $T = 22$ as well as in Shadow. Additionally, we set $noisy = 1$ in the Python simulator to capture the noise observed in our Shadow simulations. The average estimation error of *MLEFlow-Q* converged to below 10% while that of *TorFlow-P* stayed around 60% for both simulators as can be seen in figures 13a and 14a. Figures 13b and 14b show that the estimation errors of all relays have similar distribution for both simulators: the median estimation errors for *MLEFlow-Q* are around 15%, 40% and 5% for the Guard, middle and exit relays, respectively. For *TorFlow-P*, these median errors were around 50%, 100% and 30%, respectively. The distribution of bandwidth between users throughout the rounds are also similar in Python and Shadow as can be seen in figures 13c and 14c.



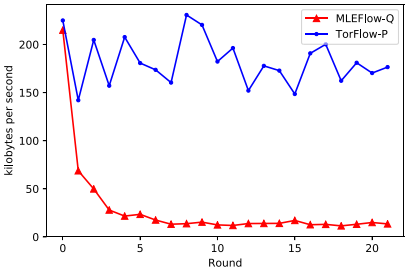
(a) Comparing average estimation errors of all exit relays in each round.



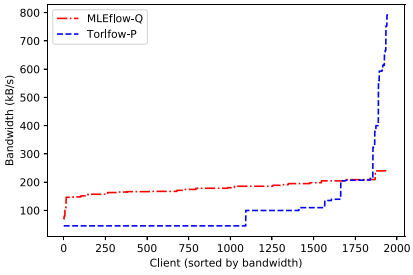
(b) Estimation error distributions for each type of relay.



(c) Aggregate mean progression of client download bandwidth distribution in each consensus round.

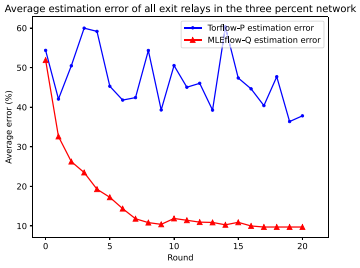


(d) Aggregate std deviation progression of client download bandwidth distribution in each consensus round.

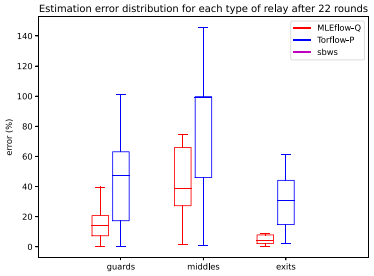


(e) Client download bandwidth distribution after bandwidth estimation has converged.

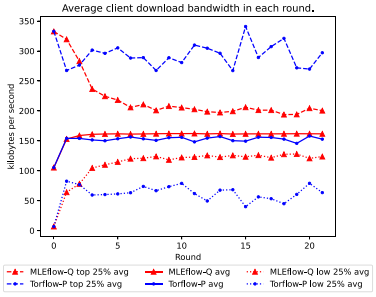
Fig. 13. Results are obtained after 22 consensus rounds in the 3% network using Python.



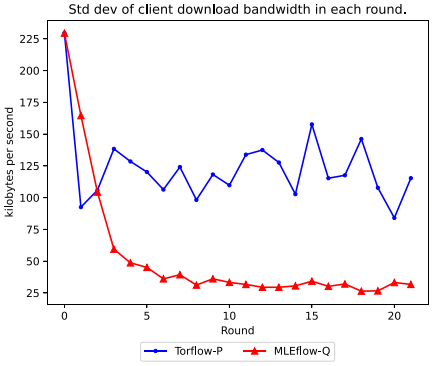
(a) Comparing average estimation errors of all exit relays in each round.



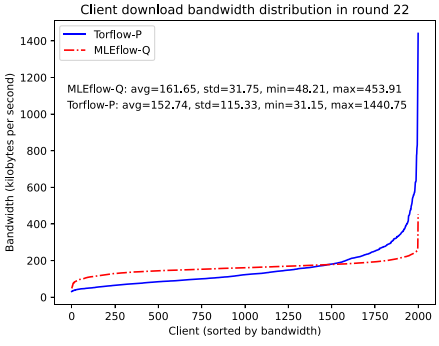
(b) Estimation error distributions for each type of relay.



(c) Aggregate mean progression of client download bandwidth distribution in each consensus round.



(d) Aggregate std deviation progression of client download bandwidth distribution in each consensus round.



(e) Client download bandwidth distribution after bandwidth estimation has converged.

Fig. 14. Results are obtained after 22 consensus rounds in the 3% network using Shadow.

C Additional Tables and Figures

Table 1. Low-fidelity simulation results done in Python3. *Est. method* is the method we use to update the weight vector in each iteration. *stats* are the statistics that we report for each experiment. *noisy* indicates if noiseless or noisy observations are considered. *relays cap. est. error* are the estimation errors for each category of relays. It is computed as follows: $error = \frac{|w_t - w^*|}{w^*} \times 100$. *paths bw* are the bandwidths allocated for users when the weight vectors are updated using different methods. The reported results are for the 50^{th} measurement period.

			Single relay paths		Three relays paths			
					relays cap. est. err. (%)			paths bw
Est. method	stats	observations noise	relays cap. est. err. (%)	paths bw (kb/s)	guard	middle	exit	(kb/s)
Actual	mean	-	0	81.48	0	0	0	22.41
	std.	-	0	6.57	0	0	0	0.77
	max	-	0	409.60	0	0	0	102.40
	min	-	0	22.94	0	0	0	8.48
Quantized	mean	-	2.4	81.48	2.4	2.4	2.3	22.41
	std.	-	1.4	6.98	1.4	1.4	1.4	0.99
	max	-	5	478.23	5	5	5	102.40
	min	-	0	18.70	0	0	0	8.19
TorFlow-P	mean	noiseless	14.30	81.47	69.22	81.86	80.25	19.76
		noisy	21.62	81.47	68.00	83.92	82.17	19.73
	std.	noiseless	20.85	9.27	63.09	79.60	47.57	174.86
		noisy	20.90	17.24	64.99	84.41	50.82	173.62
	max	noiseless	381.90	522.68	760.20	809.75	342.56	31092.96
		noisy	288.30	1048.58	1150.70	792.92	372.22	24591.96
	min	noiseless	0	17.19	0	0	0	3.30
		noisy	0	18.72	0	0	0	2.59
*sbws**	mean	noiseless	14.13	81.45	68.49	79.70	82.03	19.60
		noisy	21.79	81.47	68.47	83.11	81.54	19.63
	std.	noiseless	21.11	9.37	64.78	78.07	48.38	175.49
		noisy	21.61	17.40	64.46	83.05	51.70	171.21
	max	noiseless	385.76	524.30	874.24	1045.23	360.92	37369.87
		noisy	450.45	1024.00	1116.13	959.45	438.55	33227.31
	min	noiseless	0	14.98	0	0	0	2.66
		noisy	0	17.39	0	0	0	2.70
MLEFlow-CF	mean	noiseless	2.01	81.48	17.19	15.88	0.66	22.41
		noisy	3.29	81.48	17.23	16.20	2.21	22.41
	std.	noiseless	2.75	6.84	9.48	7.65	0.72	0.86
		noisy	3.31	7.09	10.39	8.07	1.92	1.04
	max	noiseless	24.78	559.41	44.51	36.88	8.38	51.20
		noisy	30.41	524.29	55.77	39.78	17.65	65.54
	min	noiseless	0	20.48	0	0	0	10.75
		noisy	0	17.07	0	0	0	11.42
MLEFlow-Q	mean	noiseless	4.35	81.48	16.41	17.47	2.44	22.41
		noisy	3.84	81.48	15.21	17.66	3.32	22.41
	std.	noiseless	4.75	7.33	8.95	8.58	1.67	1.10
		noisy	3.41	7.32	9.61	10.30	2.51	1.23
	max	noiseless	30.76	524.28	52.47	43.34	15.53	62.84
		noisy	26.62	614.40	49.26	57.44	18.77	51.20
	min	noiseless	0	25.60	0	0	0	11.87
		noisy	0	19.20	0	0	0	10.24

Table 2. Sensitivity of estimates to inaccurate λ_s.

Est. method	Estimated λ_s	stats	noisy	relays cap. est. err. (%) guard	middle	exit	paths bw (kb/s)
MLEFlow-CF	double	mean	0	16.38	15.77	4.24	17.74
	the		1	19.19	17.49	7.32	17.74
	actual	std.	0	12.66	10.45	3.86	1.10
	rate		1	13.44	11.6	6.27	1.69
	half	mean	0	17.29	16.32	4.2	17.74
	the		1	19.31	20.04	7.26	17.74
	actual	std.	0	12.45	11.5	3.41	1.03
	rate		1	13.22	12.08	6.19	1.71
MLEFlow-Q	double	mean	0	17.78	18.48	4.29	17.74
	the		1	19.88	23.16	6.31	17.74
	actual	std.	0	12.07	12.52	4.06	1.11
	rate		1	13.52	14.04	6.10	1.38
	half	mean	0	17.05	22.37	4.05	17.74
	the		1	19.93	23	6.27	17.74
	actual	std.	0	12.52	13.5	3.87	1.07
	rate		1	13.63	14.11	5.42	1.40

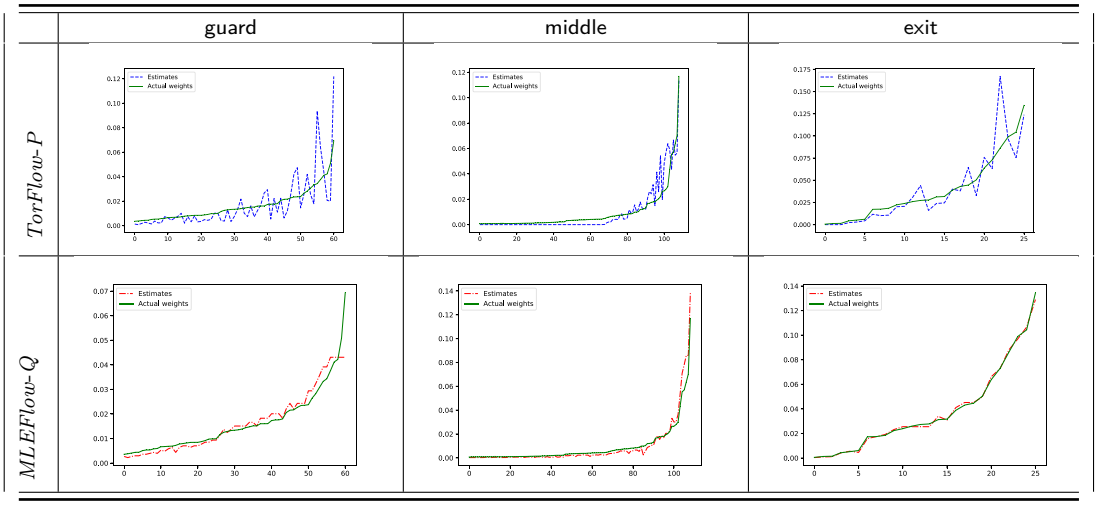


Fig. 15. Comparison of relay capacity estimates after 22 rounds of simulation in Shadow. Each subfigure shows the normalized weights of each corresponding category of relays. Each relay's estimated weight and true weight are plotted in the figure. i.e. In the *MLEFlow-Q* 3% network simulation at the exit column, 26 exit relays were plotted. The distribution of estimated weights align with the distribution of actual weights.

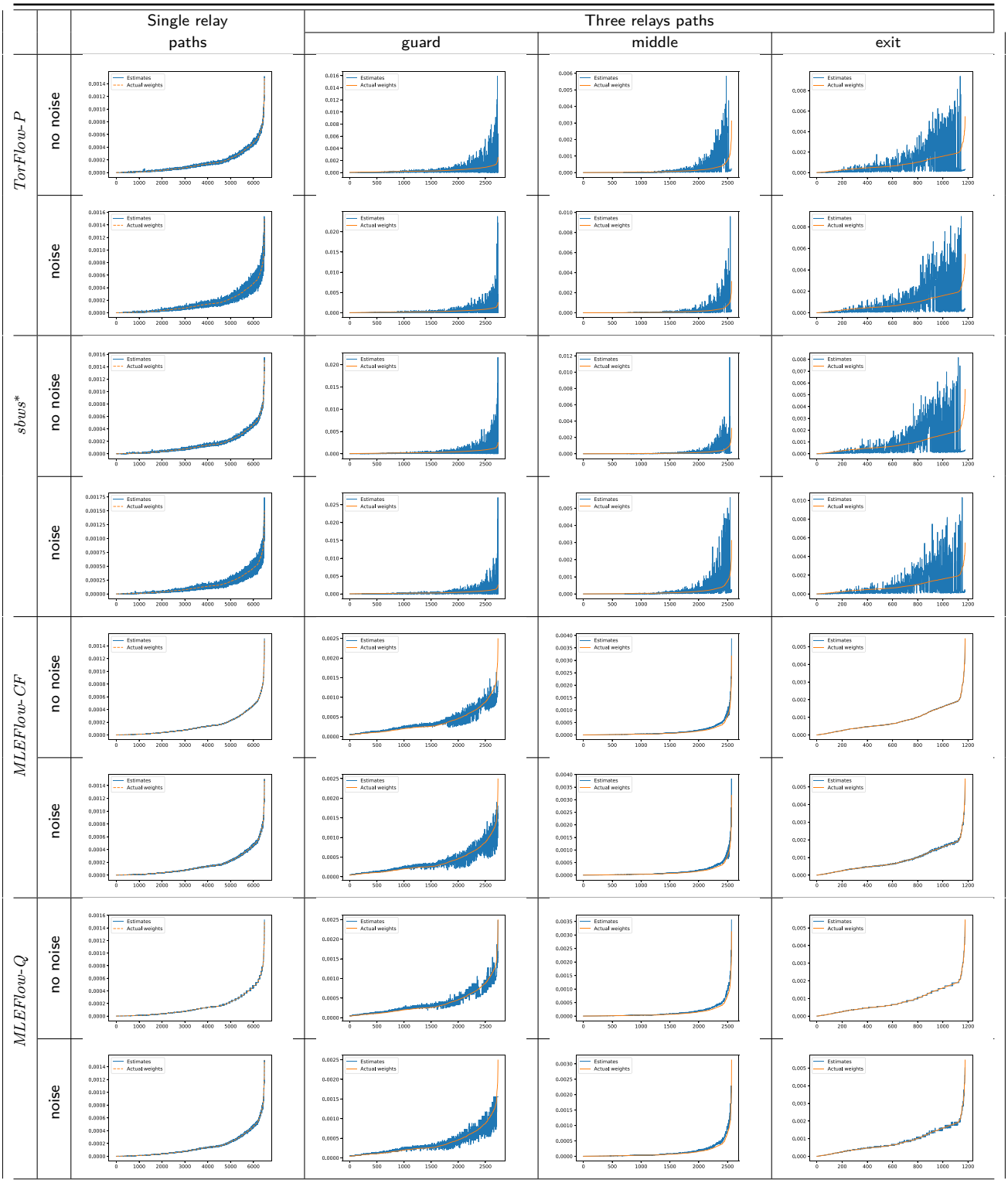


Fig. 16. Normalized relays capacities estimates using three methods for the single-relay paths and three-relay paths scenarios. The reported results are for the 50^{th} measurement period.