FRCC: Towards Provably Fair and Robust Congestion Control

Anup Agarwal, Venkat Arun†, Srinivasan Seshan
Carnegie Mellon University, †University of Texas at Austin

Abstract

Congestion control algorithms (CCAs) play a critical role in network bandwidth allocation. Recent work [3] showed that a large class of CCAs, including, BBR, Copa, and Reno, starve flows in the presence of network jitter. Starvation occurs because CCAs (implicitly) coordinate fairness by encoding fair rates into congestion signals, e.g., Reno's throughput is $\propto 1/\sqrt{\texttt{loss rate}}$. Thus, noise in congestion signals creates errors in inferring fair rates.

We present FRCC (Fair and Robust Congestion Controller) and formally prove that it converges exponentially fast to both efficient and fair rate allocations, with bounded throughput ratios even under network jitter. This is the first CCA to our knowledge to achieve this feat. Our key insight is to decouple fair rates into link capacity and flow count. We only encode flow count (or fair link fraction) into congestion signals and independently estimate link capacity, reducing jitter's impact on fairness. We implement FRCC in the Linux kernel and evaluate it in a variety of network conditions, including synthetic jitter, heterogeneous RTTs, and multi-bottleneck settings. FRCC closely matches the bounds predicted by our theoretical analysis, and consistently achieves fairness, even when state-of-the-art CCAs exhibit starvation.

1 Introduction

End-to-end congestion control algorithms (CCAs) play a critical role in allocating network bandwidth. Finding a good CCA is a challenging and long-standing problem in networking. Fairness, in particular, has been hard to achieve. While the community desires fairness across different CCAs [5, 15, 30, 35], and sophisticated notions of fairness including application objectives [10, 26, 29, 37], existing CCAs fail to provide even the simplest property: per-flow fairness *even when all flows use the same* CCA. Fig. 1 and Fig. 2 show how Cubic [16], BBR [7] and Copa [5], three widely deployed CCAs, experience extreme unfairness between flows as a result of differences in ACK aggregation or round-trip propagation delays (*RTprops*)[1]; phenomena that are common on the Internet. Appendix E shows unfairness in other CCAs.

The *starvation theorem* [3] suggests that these are not isolated failures. The theorem states that any CCA with small self-induced delay or delay variation can starve in the presence of jitter (from sources like ACK aggregation or OS scheduling delays). To our knowledge, all existing CCAs that are buffer-bloat resistant (i.e., bound delays) cause little to no self-induced delay variations and, therefore, starve. This happens on realistic paths as shown above.

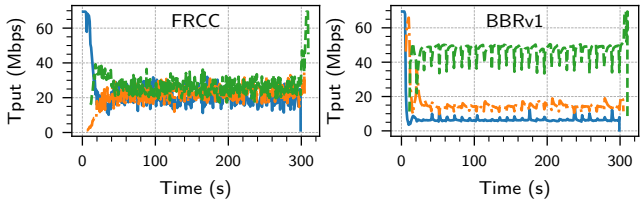

Figure 1: Throughput (Tput) of three flows with round-trip propagation delay (RTprop or R) of 10, 20 and 30 ms. BBR starves the (blue) flow with $R = 10$ ms.

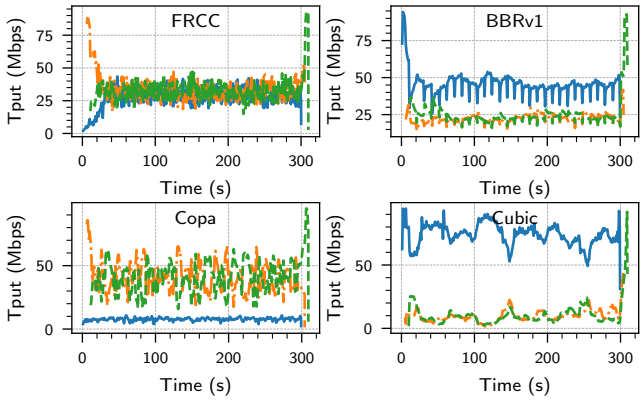

Figure 2: Three flows with RTprop = 32 ms. The blue flow experiences 32 ms of ACK aggregation. Copa and Cubic starve flows.

Given this result, we ask whether there exists an end-to-end CCA that can provably bound fairness, albeit at the cost of large (enough) self-induced delay and delay variations, while still bounding the maximum delay. We design FRCC (Fair and Robust Congestion Controller), the first such CCA with formal guarantees of fairness and efficiency under the same pessimistic, worst-case model used by [3]. With this, we can be confident that our design is robust in any real networks that can be emulated by the worst-case model, which includes components like token bucket filters and ACK aggregation.

Starvation occurs because of the way existing CCAs coordinate with other competing flows. They do not know how many other flows exist and cannot directly communicate with each other. To coordinate what is the fair rate, CCAs (implicitly) agree on a contract that encodes the fair rate into

[1]BBR's unfairness is different from RTT unfairness in traditional CCAs [16, 18]. For BBR, a small difference in RTprops leads to arbitrarily large unfairness as the link rate goes to infinity [3].

observable congestion signals. For instance, Reno uses fair rate $\propto 1/\sqrt{\texttt{loss rate}}$ [25]. Because of such coupling, even small noise in the congestion signals translates to large errors in inferring fair rates, potentially causing flows disagree on the fair rate creating unfairness. To our knowledge, all existing CCAs that share bandwidth fairly incorporate such a map in their design, either implicitly (e.g., Reno [25], Vegas [24]), or explicitly (e.g., TFRC [12], Swift [21], Poseidon [34]).

FRCC incorporates two key ideas to enable fairness in the presence of jitter. First, rather than coordinating fair *rates* through congestion signals, FRCC coordinates flow count or fair link *fractions*. Flows independently estimate the bottleneck link capacity to know if the fraction of link they *currently* consume is more or less than the *target* fraction encoded in the congestion signals. Flows adjust their `cwnd` to consume the same fraction of the link as communicated in the congestion signals. This reduces the bits of information communicated through congestion signals, thereby reducing the impact of noise.

Second, capacity estimation is notoriously hard ([11], § 5.3), often requiring large probes for high accuracy. Our key idea is that FRCC only needs the capacity estimation to be accurate enough for flows to know if their current link fraction is above or below the target fraction. We precisely compute how large and long our probes need to be to build an accurate enough capacity estimator. This creates the delay variations that we know are necessary to avoid starvation from [3].

Our proofs model the execution of FRCC as a dynamical system, describing both its transient and steady-state performance (§ 6). We cover a range of scenarios from ideal links (without jitter), links with jitter, and even flows vastly different RTprops and multiple-bottleneck links. To exhaustively reason about network behaviors, we use a number of analysis tools, including the Z3 SMT (satisfiability modulo theory) solver [27], and numerical methods [32]. Our formal approach also uncovered previously undocumented behaviors in `cwnd`-based CCAs that occur due to ACK-clocking (§ 5.3.2) and are unrelated to jitter. These are interesting in their own right independent of FRCC.

We implement FRCC in the Linux kernel and empirically evaluate it. FRCC behavior closely matches our theoretical analysis for all the challenging scenarios listed above and consistently achieves fairness and efficiency, even when state-of-the-art CCAs exhibit starvation (e.g., Fig. 1, Fig. 2).

Note that our goal was to investigate if it is possible to design a CCA that operates at the limits of existing theoretical tradeoffs. FRCC's current design does not consider important practical issues such as coexistence with other CCAs, shallow-buffers, etc. (§ 2). Further, we had to make several assumptions (listed in Appendix C) to make our proofs tractable. We conjecture that our results are true even without these assumptions as supported by our empirical evaluation (§ 7).

We have open sourced our code at `https://github.com/108anup/frcc/tree/main` described in Appendix A.

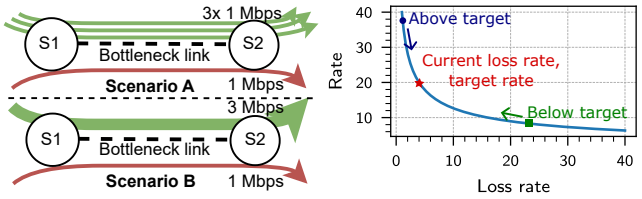

Figure 3: [Left] To distinguish between the two scenarios, the red (lower) flow must coordinate with the green (upper) flows. **[Right]** Reno uses loss rate to communicate fair rates.

2 Related work and motivation

Starvation theorem statement [3]. On networks with jitter, no end-to-end CCA can simultaneously achieve three desirable properties: (P1) avoid starvation, i.e., ensure a finite bound on the ratio of flow throughputs (s-fairness in [3]), (P2) scale to arbitrarily large link capacities (f-efficiency in [3]), and (P3) incur only small self-induced delay or delay variation (delay convergence in [3]).

This result applies only to end-to-end schemes, not to those relying on in-network support (e.g., XCP [20]) or active queue management (e.g. fair queuing). We focus on end-to-end CCAs because in-network mechanisms have been historically hard to deploy: explicit schemes require header changes (via IP or TCP options), which risk packet drops at legacy routers and incompatibility with encryption [14].

Goals. Our motivation is to address the practical starvation observed in Fig. 1 and Fig. 2, both of which arise in common scenarios. For example, two users in a household may have different RTprops when downloading from separate CDNs (e.g., YouTube and Amazon CloudFront, both using BBR), causing one flow to starve. Likewise, when one user connects via Ethernet and another via WiFi, the resulting differences in jitter lead to starvation under Cubic, Reno, or Copa. In the context of the starvation theorem, rectifying these issues requires achieving both (P1) bounded fairness and (P2) scaling to large capacities (e.g., 100 Mbps in Fig. 1 and Fig. 2). So we must give up (P3).

Consequently, our main goal was to determine if an end-to-end CCA can provably avoid starvation (achieve bounded fairness) and scale to arbitrarily large link capacities in the presence of jitter by accepting the tradeoff of creating large self-induced delays and delay variations.

Here, large is a relative term, and the absolute value of delay and delay variations can still be low. Specifically, these only need to be (1) larger than the amount of jitter in the network, and (2) be *independent of the link capacity*. CCAs, like Vegas/Copa/BBR, may maintain large delays or delay variations, but these vanish with increasing link capacity (§ 7), and their unfairness increases with link capacity, eventually leading to starvation. This also means that we can still bound the maximum delay and delay variation we create to avoid buffer-bloat, unlike traditional CCAs like Reno and Cubic.

This represents a challenging theoretical problem. To date, no delay-bounding end-to-end CCA has been proven to

avoid starvation, particularly when faced with arbitrary jitter patterns as modeled by the CBR-delay network model (Fig. 13 and [3]). While these jitter patterns may seem pessimistic or rare, existing CCAs break under simple scenarios (Fig. 1, Fig. 2). We believe that designing FRCC for the pessimistic model, even though it may not capture all real-world scenarios, will make it robust to a wide range of network behaviors.

Non-goals. Beyond our primary objectives of bounded fairness, high utilization (scaling to arbitrarily large bandwidths), and bounded maximum delay (and delay variations), our secondary goal was to design a practical CCA. We partially achieve this goal. FRCC converges exponentially fast to both efficient and fair rate allocations (its convergence time is $O(N_G \log \texttt{BDP})$ RTTs), where N_G is the number of flows.

We do not address other properties that may be desirable for a CCA, including, coexistence with other CCAs, operation under shallow buffers, handling losses (including detection, recovery, and reaction), or short/application-limited flows.

Our analysis assumes buffers are sufficiently large to avoid loss. Here, sufficiently large is relative to the amount of jitter and not RTprop. Specifically, FRCC maintains a steady-state queueing delay of $O(N_G \mathcal{D})$ (§ 6). Where, $\mathcal{D}$ is the amount of jitter we want FRCC to tolerate. If RTprop $=$ 100 ms, $N_G = 4$ flows, and $\mathcal{D} = 10$ ms, FRCC needs $4 \times 10 = 40$ ms of buffering. In contrast, CCAs like Reno/Cubic/BBR need $O(\text{RTprop})$ or 100 ms of buffering. We recently showed that buffering smaller than jitter is also an extremely challenging scenario involving an additional tradeoff between amount of packet loss and convergence time [1].

Similarly, our community has luxurious desires from CCAs, including coexistence with legacy TCP [5, 15, 30, 35] and sophisticated fairness notions based on latency or application-visible metrics [10, 26, 29, 37]. However, today's CCAs fail to ensure even basic flow-level throughput fairness on simple dumbbell topologies *where all flows are using the same CCA*. We need to solve this before pursuing higher ambitions.

Contracts. The starvation result stems from the way CCAs coordinate fair rates with each other. CCAs do not know the number of competing flows. To determine their fair rate, they must establish some form of agreement. Consider the scenarios in Fig. 3. If the green (upper) flow in scenario B exactly emulates the cumulative effect of the three green flows in scenario A, then the red flow cannot tell the difference. Without disambiguating the scenarios, the flow cannot have different actions (sending rates) in the two scenarios. Yet, its fair rate is different (1 and 2 Mbps in scenario A and B, respectively).

In practice, the red flow assumes the green flow is cooperating through a "contract"—an a priori agreed-upon encoding of fair rates into observable network signals. For example, Reno uses the contract function "$\texttt{rate} = \texttt{const}/\sqrt{\texttt{loss rate}}$" [12, 25]. While this is not explicit in the algorithm, the emergent behavior is that all flows measure the average packet loss rate and calculate a "target rate" using this formula. Then they update their actual sending rate to move towards this target

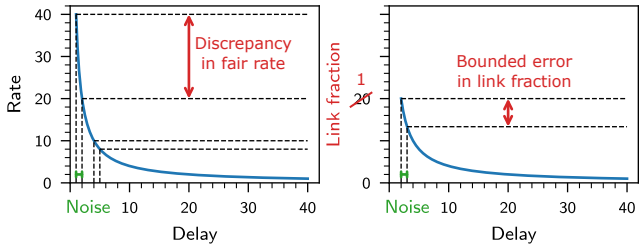

Figure 4: [Left] Noise creates discrepancies in inferring fair rates. The discrepancy increases with increasing link capacity. **[Right]** Link fraction gets rid of the asymptote on the Y-axis, ensuring bounded error in inferring fair link fraction for bounded jitter.

(Fig. 3). Returning to scenario B from Fig. 3, both the red and the green flows measure the same loss rate, and hence, have the same target rate, but they have different sending rates. Thus, at least one of them will adjust their rate until they reach equal sending rates in steady-state.

To our knowledge, all existing CCAs employ similar coordination methods, though the signals they use and the shape of the contract function may vary. E.g., Vegas, FAST, Copa, and Swift, use delay instead of loss [5, 6, 21].

Why does the starvation occur? The issue with this coordination is that rate gets coupled with noise in congestion signals. The contract maps a small range of congestion signals to a large range of rates. As a result, even small noise in congestion signals creates large errors in inferring fair rates, causing different flows to disagree on what is the fair rate (Fig. 4). This disagreement increases with increasing link capacity, and due to the asymptote on Y-axis, creates arbitrarily large ratios of flow throughputs (starvation) for arbitrarily large capacity.

3 Key insights to overcome starvation

Why do strawman solutions fail and key learnings. From Fig. 3, to coordinate fairness, we need some form of contract (e.g., rate-vs-delay mapping). And, the asymptote in such contracts is the main culprit causing starvation (Fig. 4). The problem is that there is no rate-vs-delay curve that (1) is asymptote-free, (2) scales to arbitrarily large rates, and (3) is monotonically decreasing. Monotonic decrease is needed as increasing rate with delay creates a positive feedback loop.

This explains why decades of research have failed to resolve starvation: no amount of noise filtering or parameter tuning can eliminate the asymptote. At high link capacities, even microsecond-level jitter causes large divergence in the inferred fair rates. Adjusting parameters only shifts the asymptote rather than removing it. For example, Vegas uses the contract $\texttt{rate} = \alpha_{\texttt{Vegas}}/\texttt{delay}$. Increasing $\alpha_{\texttt{Vegas}}$ allows Vegas to tolerate up to 10 ms of jitter on a 100 Mbps link by maintaining a 10 ms queueing delay. However, this tuning dramatically increases delay at lower capacities—e.g., 1 s of queueing at 1 Mbps—and the asymptote resurfaces at higher capacities (e.g., 1 Gbps). Similarly, changing the contract shape (e.g., $\texttt{rate} = 1/\texttt{delay}^2$ [21]) does not eliminate the asymptote, and

contract proposals such as "$\texttt{rate} = C_{max}e^{-\texttt{delay}}$" [3] only eliminate the asymptote by give up scaling to arbitrary bandwidths.

First key insight. Instead of relying on a contract that coordinates the fair *rate* across flows, we split the challenge for determining fair rate into two parts. Our first insight is that inter-flow coordination should focus on the *fair link fraction*, i.e., the fraction of the bottleneck link that flows should occupy in steady-state. For example, we use the contract: "$\texttt{target\_link\_fraction} = \min(1, \mathcal{D}/\texttt{delay})$" (shown in Fig. 4). This eliminates the asymptote in the contract since the link fraction is naturally bounded by 1. Consequently, bounded noise in delay measurements creates bounded errors in inferring the fair link fraction. Here, $\mathcal{D}$ is a tunable bound on the amount of network jitter that we want to tolerate.

Each flow updates its `cwnd` to ensure that the fraction of the link it consumes is equal to the globally observable target fraction, so that in steady-state, all flows occupy the same fraction of the link (fairness). Flows increase their `cwnd` if their current fraction is less than the target and decrease otherwise.

Second key insight. The above requires that flows estimate the fraction of the link they currently consume. One way to achieve this is by estimating their throughput and the bottleneck link capacity, i.e., $\texttt{current\_fraction} = \texttt{throughput}/\texttt{capacity}$. To estimate capacity, flows can "probe" the link by temporarily increasing their `cwnd`. However, estimating link capacity is a surprisingly difficult problem [11, 22, 23], with accurate estimates requiring large probes (§ 5.3). Our second key insight is that our capacity estimates only need to be accurate enough for flows to determine if their current fraction is above or below the target so that they can change their `cwnd` in the correct direction.

We precisely compute how large capacity probes need to be, allowing us to make our probes, and hence, delay variation smaller than otherwise needed (§ 5.3). Specifically, our probes increase the `cwnd` by:

$$E = \frac{\texttt{throughput}}{\texttt{target\_fraction}} \cdot \gamma\mathcal{D} \tag{3.1}$$

where $\gamma > 1$ is a constant. This probe increases queuing delay by E/C seconds, i.e., all packets packets have to wait behind (the transmission delay of) the increased inflight. The starvation theorem does not apply to this design due to the resulting delay variation.

This probe is large enough for us to compare the target and current fraction. The intuition is that when the target and current fractions are close (their ratio is close to 1), the flow's $\texttt{throughput} \approx \texttt{target\_fraction} * C$. As a result, the probe size (Eq. 3.1) is close to $C\gamma\mathcal{D}$, and creates $\gamma\mathcal{D}$ delay. Jitter can only impact the expected delay by $\pm\mathcal{D}$. If the measured delay is more than $\gamma\mathcal{D}+\mathcal{D}$, then the probe must have created more than $\gamma\mathcal{D}$ delay, which can only happen if the flow's throughput is larger than the target fraction of the link. Conversely, if the excess delay is less than $\gamma\mathcal{D}-\mathcal{D}$, flow's throughput is smaller. When all flows' excess delays are close to $\gamma\mathcal{D}$, they consume a fraction of the link that is close to the target fraction (fairness).

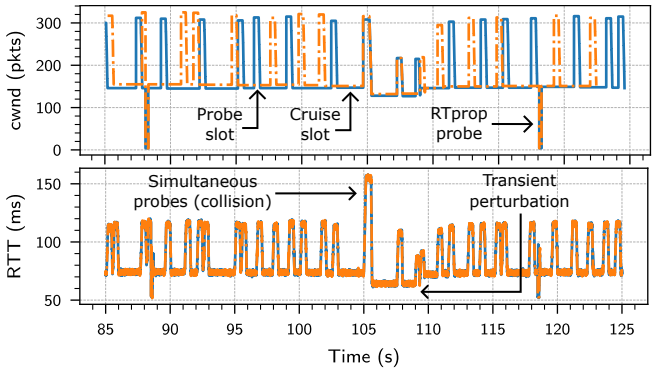

Figure 5: Timeseries of FRCC's `cwnd` and RTT.

Summary. We use delays larger than jitter to (accurately enough) coordinate fair link fractions, and delay variations larger than jitter to (accurately enough) estimate link capacity (and current link fraction). These allow us to ensure that the link fractions of all flows (and correspondingly the throughputs of the flows) are close to each other (fairness).

3.1 Leveraging the insights

We combine these insights to build a practical congestion control loop. For reference, Alg. 2 in § B.1 provides FRCC's pseudocode. For convenience in implementation and analysis, we perform a change of variables:

$$\texttt{current\_flow\_count} = 1/\texttt{current\_link\_fraction}$$
$$\texttt{target\_flow\_count} = 1/\texttt{target\_link\_fraction}$$

FRCC tracks estimates for *four independent quantities*: (1) RTT ($\widehat{\texttt{rRTT}}$), (2) RTprop ($\widehat{R}$), (2) throughput ($\widehat{\texttt{rtput}}$), and (4) link capacity ($\widehat{C}$), where $\widehat{\texttt{rRTT}} - \widehat{R}$ gives `delay`. These give us the current ($\widehat{N_C}$) and target ($\widehat{N_T}$) flow counts as:

$$\widehat{N_C} = \frac{\widehat{C}}{\widehat{\texttt{rtput}}} \quad \text{and} \quad \widehat{N_T} = \texttt{Contract}_{\texttt{Func}}(\widehat{\texttt{rRTT}} - \widehat{R}) \tag{3.2}$$

Where $\texttt{Contract}_{\texttt{Func}}$ is our contract function (§ 5.1).

Since we cannot simultaneously measure all four quantities [7], we use a structure similar to BBR [7] to obtain estimates over time (Fig. 5). We divide time into rounds, which are further divided into slots. There are two types of slots: (1) probe and (2) cruise. Each flow independently selects exactly one slot in the round as its probe slot. Flows make this choice uniformly randomly and freshly for each round.

In the probe slot, FRCC temporarily increases `cwnd` to estimate link capacity ($\widehat{C}$). In the cruise slot, FRCC retains its `cwnd` to measure throughput ($\widehat{\texttt{rtput}}$) and RTT ($\widehat{\texttt{rRTT}}$). In addition to the slots, every $T_R = 30$ seconds, FRCC flows launch a synchronized RTprop probe to empty queues and estimate RTprop ($\widehat{R}$), similar to BBR [7]. To ensure synchronized draining, we assume clock synchronization (through the Network Time Protocol) to keep implementation simple, though we could use BBR's implicit synchronization as well (§ B.4). Finally, at the end of the probe slot, FRCC updates `cwnd` to move the current and target flow counts closer to each other.

Note, we use a `cwnd` instead of rate to keep a closed-loop system and bound queues. We pace packets at the rate of $2 \cdot \texttt{cwnd}/\widehat{R}$ to avoid transmitting bursts while ensuring we do not get rate-limited. This is a common-case optimization and does not affect performance under worst-case jitter.

4 Design requirements

Notation. We use hats to denote estimates (e.g., $\widehat{N_{C_i}}$, $\widehat{N_{T_i}}$) and no annotation to denote true values (e.g., N_{C_i}, N_T), and subscript of i for flow f_i. We drop the subscript when clear from context. You can find our glossary in Table 1 and Table 2.

Desired bounds. Flows update their `cwnd` based on their local estimates, $\widehat{N_{T_i}}$ and $\widehat{N_{C_i}}$. To ensure throughput ratios are bounded, we need to ensure: **(D1)** the target flow counts of all flows are within a multiplicative bound of each other, and **(D2)** each flow is able to infer when its true current flow count (N_{C_i}) (inverse link fraction) is multiplicatively far from its local target ($\widehat{N_{T_i}}$). On combining these, the true current flow count (inverse link fraction) N_{C_i} is close to local target ($\widehat{N_{T_i}}$), which is close to the single true global target (N_T). So the true link fractions (N_{C_i}) (and thus throughputs) are close to each other.

Meeting D1/D2. From Eq. 3.2, $\widehat{C}$ and $\widehat{\texttt{rtput}}$ affects error in $\widehat{N_C}$ and the shape of our $\texttt{Contract}_{\texttt{Func}}$ affects how errors in delay ($\widehat{\texttt{rRTT}} - \widehat{R}$) propagate to $\widehat{N_T}$ (D1). Finally, errors in both $\widehat{N_C}$ and $\widehat{N_T}$ affect (D2). It is challenging to directly reason about necessary error bounds on the four individual estimates to meet (D1/D2). We build $\widehat{C}$ using our second key insight (§ 3) and we build other estimates $\widehat{\texttt{rRTT}}$, $\widehat{R}$, and $\widehat{\texttt{rtput}}$ to be the best achievable under our error model below. We then show that our estimates are indeed sufficient to meet (D1/D2). We do this by encoding all the error bounds in the Z3 SMT solver [27] and show that FRCC converges to bound fairness (§ 6.3).

Error model. We need to consider three sources of errors: **(E1)** network jitter, **(E2)** RTT (and throughput) variations due to a combination of multiple hops (§ 5.3), difference in feedback delay (RTprop) of flows (§ 5.3), and interleaving of packets across flows (§ B.3), and **(E3)** self-induced delay variations due to our capacity probes.

Tolerating E1/E2. (E2) errors are unrelated to network jitter and even occur on smooth, non-time varying links. In two instances, we were able to model these and mitigate their impact (§ 5.3, § B.3). Outside of these two instances, we treat (E2) same as (E1). We design FRCC to tolerate a bounded (configurable) amount of cumulative errors due to (E1) and (E2). We model these errors as non-deterministic (or arbitrary) similar to prior work [1, 3, 4], as opposed to stochastic (or random). Arbitrary delays (as opposed to stochastic delays) can express causal effects and correlations, allowing us to be provably robust to a number of sources of jitter in real networks [4].

Tolerating E3. (E3) affect us in two ways. First, when a flow probes, it temporarily inflates RTT and reduces throughput for other flows, potentially skewing their estimates of $\widehat{\texttt{rRTT}}$, $\widehat{R}$, and $\widehat{\texttt{rtput}}$. Fortunately, these are one-sided errors: RTTs only increase and throughputs only decrease. We filter them by computing $\widehat{\texttt{rRTT}}$ and $\widehat{R}$ as the min over RTT samples, and $\widehat{\texttt{rtput}}$ as the max over throughput samples collected in cruise slots. This bounds additive error in $\widehat{\texttt{rRTT}}$ (best achievable under (E1/E2)). For $\widehat{\texttt{rtput}}$, we measure throughput over packet-timed RTT intervals (i.e., the interval starts with sending a packet and ends with receiving its ACK) which bounds multiplicative error (also the best achievable [1]). We show in § B.3 that measuring throughput over other (even potentially longer) intervals is either sub-optimal or provides no additional benefit.

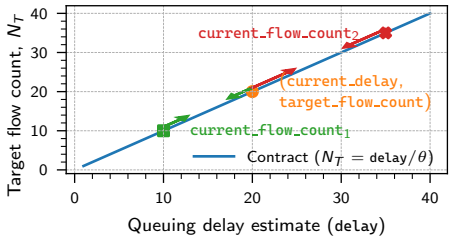

Figure 6: Contract used by FRCC. FRCC updates `cwnd` so that the target and estimated flow counts move towards each other.

Second, if two flows probe for capacity together, their induced delays add up and they misestimate capacity ($\widehat{C}$). Our round and slot design is specifically to address the second issue. Probing in exactly one slot of the round reduces the likelihood of simultaneous probes from multiple flows (§ 5.4) and helps retain fairness despite probe collisions.

5 Design

We begin by describing FRCC's contract which governs its steady-state performance (§ 5.1). Then we describe how we perform `cwnd` updates to reach this steady-state (§ 5.2). The remaining subsections detail the design of our estimates and slots (§ 5.3, § 5.4, § B.4). In these, tolerating the error bounds directly guides our design of estimates: $\widehat{N_T}$ (§ 5.1), $\widehat{C}$ (and $\widehat{N_C}$) (§ 5.3 for (E1/E2), and § 5.4 for (E3)). We covered $\widehat{\texttt{rRTT}}$ and $\widehat{\texttt{rtput}}$ in § 4, and $\widehat{R}$ in § 3 and § B.4.

5.1 FRCC's Contract

We use the contract (Fig. 6):

$$\texttt{Flow count} = \texttt{Contract}_{\texttt{Func}}(\texttt{delay}) = \texttt{delay}/\theta \qquad (5.1)$$

We interpret it in two ways: (1) "**target flow count** as a function of **current delay**" ($N_T = \texttt{delay}/\theta$), and (2) "**target delay** as a function of **current flow count**" ($\texttt{delay} = \theta \cdot N_C$). The constant $\theta > 0$ denotes the seconds of delay we maintain per flow in steady-state, and amount of noise we tolerate: ΔR of noise in `delay` creates $\Delta R/\theta$ of additive error in $\widehat{N_T}$. Increasing θ increases error tolerance at the expense of delay. In steady-state, without jitter, $N_T = N_C = N_G$, where N_G is the ground truth number of competing flows (§ 6.2). I.e., all link fractions (and throughputs) are equal and there is θN_G of queuing delay.

Past CCAs have employed different contract shapes, e.g., linear (Copa/Vegas/FAST, $\texttt{delay} = 1/\texttt{rate}$ [5, 6, 36]), square-

root (Swift, $\texttt{delay} = 1/\sqrt{\texttt{rate}}$ [21]), square (Reno [18, 25], $\texttt{loss\_rate} = 1/\texttt{rate}^2$). The shape affects three metrics [2]: (M1) delay growth with flow count, (M2) tolerance to jitter, and (M3) fairness with multiple bottlenecks (e.g., Reno achieves minimum potential delay fairness, while Copa/Vegas/FAST achieves proportional fairness [31]). Since a single choice affects all these metrics, they are at odds with each other [2].

While our contract bounds additive error in $\widehat{N_T}$, we only needed to bound multiplicative errors (for D1 in § 4). The logarithmic contract, $\texttt{delay} = \theta\log(N_C)$, gives the optimal tradeoff between delay (M1) and noise tolerance (M2) [2, 3]. Delay grows logarithmically with flow count and multiplicative error in $\widehat{N_T}$ is bounded ($\widehat{N_T} = e^{(\texttt{delay}\pm\Delta R)/\theta} = e^{\pm\Delta R/\theta}\cdot N_T$). However, this creates unfairness with multiple bottlenecks (M3) (§ 6.5), so we use do not use this contract.

5.2 Congestion window update

At the end of the probing slot, flows update their `cwnd` to move the current and target flow counts closer to each other (Fig. 6). When current exceeds target ($\widehat{N_C} > \widehat{N_T}$), FRCC increases `cwnd`. This decreases the current flow count (increases link fraction), and increases the target (due to the increased delay). Likewise, when current is below target ($\widehat{N_C} < \widehat{N_T}$), FRCC decreases `cwnd`.

The `cwnd` updates are proportional to the gap between $\widehat{N_C}$ and $\widehat{N_T}$. This enables: (1) exponentially fast convergence to both efficiency (link utilization) and fairness in $O(\log \texttt{BDP})$ rounds, and (2) stability, as proportional updates dampen as FRCC gets closer to steady-state, similar to [17, 36].

Algorithm 1: FRCC `cwnd` update.

1 **Function** `updateCwnd()`
2 $\quad\widehat{N_T} \leftarrow (\widehat{\texttt{rRTT}} - \widehat{R})/\theta$ ▷ `cwnd` $\leftarrow$ `cwnd`δ_H if $\widehat{N_T} = 0$
3 $\quad\widehat{C} \leftarrow E/\widehat{\Delta d}$ ▷ $\widehat{C} \leftarrow \infty$ if $\widehat{\Delta d} \le 0$ (See § 5.3)
4 $\quad\widehat{N_C} \leftarrow \max(\widehat{C}/\widehat{\texttt{rtput}}, 1)$ ▷ As $\texttt{rtput} \le C$
5 $\quad\widehat{N_C} \leftarrow \min(\max(\widehat{N_C}, \widehat{N_T}/\delta_L), \widehat{N_T}\delta_H)$ ▷ Clamps
6 $\quad\texttt{target\_cwnd} \leftarrow \texttt{cwnd}\cdot\widehat{N_C}/\widehat{N_T}$ ▷ Alternate `cwnd` update
7 $\quad\texttt{target\_cwnd} \leftarrow \texttt{cwnd}\cdot(\widehat{R} + \theta\widehat{N_C})/(\widehat{R} + \theta\widehat{N_T})$
8 $\quad\texttt{cwnd} \leftarrow (1-\alpha)\cdot\texttt{cwnd} + \alpha\cdot\texttt{target\_cwnd}$ ▷ $\alpha = 1$
9 $\quad\texttt{cwnd} \leftarrow \textbf{ceil}(\texttt{cwnd})$ ▷ Ensures at least 1 packet inflight
10 **end**

Alg. 1 shows two `cwnd` updates: line 7 (main) and line 6 (alternate). The main update is more stable and we use it in all our analysis and implementation. We only use the alternate update in our proof for the jittery link in § 6.3 as Z3 times out for the main update (**Assumption 6.**).

The main update focuses on moving only the "queuing delay" portion of the RTT towards our target delay from $\theta\widehat{N_T}$ to $\theta\widehat{N_C}$. Intuitively, `cwnd` maps to "packets in queue + packets in pipe". The main update isolates and scales the term for "packets in queue", i.e., it is same as: $\texttt{target\_cwnd} = \texttt{rate} * \texttt{target\_RTT} = \texttt{cwnd}/\widehat{\texttt{rRTT}} * (\widehat{R} + \theta\widehat{N_C})$. In contrast, the alternate update is more aggressive (less stable) and scales the entire `cwnd`. Specifically, the alternate update requires $\theta N_G > R$ for stability, i.e., the queuing portion of the `cwnd` is more dominant than the pipe portion.

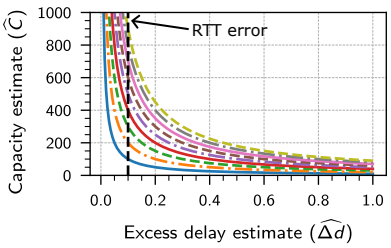

Figure 7: Lines show increasing E. A small range of excess delays map to a large range of potential link capacities. For $C = 600$, only the largest three probes create more than ΔR of excess delay.

Clamps on `cwnd`. The clamps (line 4 and line 5) help manage the impact of noise in estimates in three ways. First, while without jitter, FRCC converges to a fixed-point, with (worst-case) jitter, FRCC converges to a region around the fixed-point (§ 6.3). The clamps (among other parameters, and the amount of jitter) affect the size of the region. Tighter clamps imply a smaller region but slower convergence. Second, our capacity estimate is more accurate for some flows than others (§ 5.3.1). With worst-case errors, some flows may move away from the steady-state region. The clamps curb these bad movements. The onus is on the flows with accurate estimates to bring the system to convergence. Lastly, clamps help curb perturbations when FRCC misestimates capacity due to probe collisions.

5.3 Link capacity estimate

Prior work has explored various capacity estimation techniques, including packet-pairs/packet-trains [22, 23] and max ACK rate [7, 15]. All these methods have known issues, e.g., instead of estimating capacity, max ACK rate and packet-trains may only measure available bandwidth and asymptotic dispersion rate [11] when other flows consume a part of the link.

The three sources of errors (E1/E2/E3) make capacity estimation hard. We elaborate why and describe our solutions. Specifically, for (E1) we ensure that our capacity probes are large enough (§ 5.3.1), for (E2) we measure persistent change in delay created by the probes as opposed to transient change (§ 5.3.2) and for (E3) we have our slot/round design (§ 5.4).

5.3.1 Probe size

Problem (Fig. 7). Consider the simple estimator we used in § 3. It increases `cwnd` by E packets, and measures the excess delay created by the probe ($\widehat{\Delta d} = \Delta RTT$) to estimate capacity as: $\widehat{C} = E/\widehat{\Delta d}$. When there is no noise, the estimator is correct (as $\widehat{\Delta d} = \Delta d = E/C$). When there is noise, i.e., $\widehat{\Delta d} = \Delta d \pm \Delta R$, this estimator has a similar issue as we saw in Fig. 4. It maps a small range of $\widehat{\Delta d}$ values to a large range of $\widehat{C}$ values (Fig. 7). Small noise in $\widehat{\Delta d}$ ($= \Delta d \pm \Delta R$) creates large errors in $\widehat{C}$. The same issue happens for an estimator based on the increase in throughput (or ACK rate) due to probe.

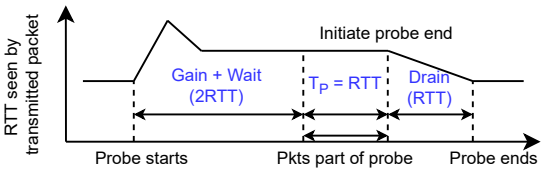

Figure 8: Life-cycle of a probe slot.

Unless $E > C\Delta R$, the excess delay is less than jitter, and the CCA cannot disambiguate between queueing delay and noise. C is the quantity we are trying to estimate, so we cannot set such a probe size a priori, to bound the error in $\widehat{C}$ and consequently $\widehat{N_C}$. Though it may be possible to binary search for such a probe size on the fly.

Solution. We only need to decipher if our current flow count is above or below target. For this purpose $E = \widehat{N_T}\widehat{\texttt{rtput}}\cdot\gamma\mathcal{D}$ is enough. Under this probe size, if we propagate the impact of errors from $\widehat{\Delta d}$ to $\widehat{N_C}$, we get (derivation in § B.2):

$$\frac{\widehat{N_T}}{\widehat{N_C}} = \frac{\widehat{\texttt{rtput}}}{\texttt{rtput}}\frac{\widehat{N_T}}{N_C} \pm \frac{\Delta R}{\gamma\mathcal{D}} = \gamma_R\frac{\widehat{N_T}}{N_C} \pm \frac{1}{\gamma} \tag{5.2}$$

Here, $\texttt{rtput}$ and N_C are the true throughputs and current flow count (inverse fraction) of the flow, γ_R is the multiplicative error in throughput, and $\gamma\mathcal{D}$ controls the tradeoff between the seconds of delay variation we create, and the accuracy of our estimate. We get the second equality whenever $\Delta R \leq \mathcal{D}$. Thus, the total error in $\widehat{N_T}/\widehat{N_C}$ is bounded and the flows update $\texttt{cwnd}$ in the correct direction when $\widehat{N_T}$ and N_C are far (D2).

Note, the additive error in $\widehat{N_T}/\widehat{N_C}$ causes some flows to have more accurate estimates than others. This asymmetry occurs in two ways. First, the estimate is more accurate for flows whose $\widehat{N_T}$ and N_C are far apart ($\widehat{N_T}/N_C$ is too large or too small). Second, when $\widehat{N_T}/N_C \in (0,1)$, i.e., additive error has greater impact than when $\widehat{N_T}/N_C \in (1,\infty)$. In essence, flows getting more than their fair share have better estimates. While not all flows may move towards the steady-state region in a given round, flows that are too fast or too far from the fair share are more likely to move in the correct direction and update the global N_T to help bring the system to convergence.

Note, bounding multiplicative error in $\widehat{N_T}/\widehat{N_C}$ relative to $\widehat{N_T}/N_C$, to avoid asymmetries, is same bounding error in $\widehat{N_C}$ which requires $O(C)$ probe size as shown above.

5.3.2 Probe timing

With different RTprops, or multiple hops (either bottleneck or non-bottleneck), probes incur transient variations in RTT (and throughput) even without jitter. Bottleneck hops are those that have other competing FRCC flows (Fig. 17), while non-bottleneck hops do not have any competing flows (Fig. 9). With different RTprops and multiple *bottleneck* hops there is also *persistent* bias in excess delay, i.e., the *ground truth* $\Delta d \neq E/C$. We explain the cause and impact of persistent bias in § 6.5. Here, we discuss the transient variations. Both the

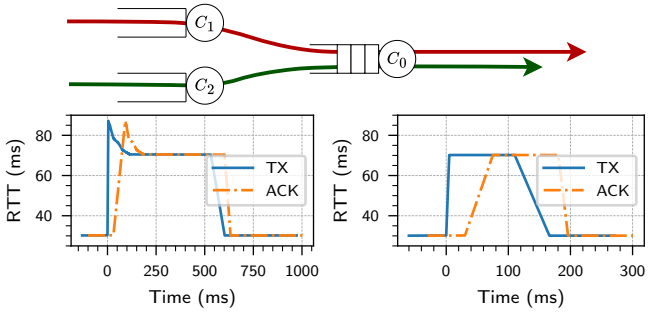

Figure 9: Packet trains may not measure the true link capacity. TX shows RTT of a packet vs. its transmission time, while ACK shows the RTT of a packet vs. its ACK time. **[Top]** C_1 and C_2 are non-bottleneck hops, C_0 is the bottleneck hop. Behavior in left plot occurs whenever $C_1, C_2 \geq C_0/2$. **[Left]** with extra hop (e.g., C_1, C_2), **[Right]** with only the bottleneck hop.

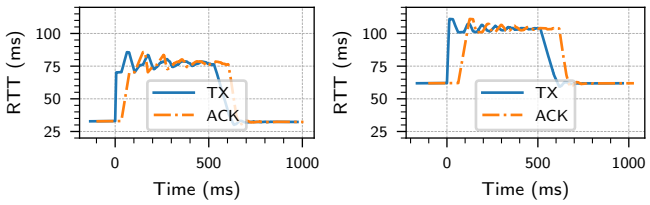

Figure 10: Oscillations occur during probes when flows have different RTprops. The excess delay without oscillations $\neq$ the transmission delay of excess packets. Oscillations dampen over time. Left shows probe of short RTprop flow and right shows long.

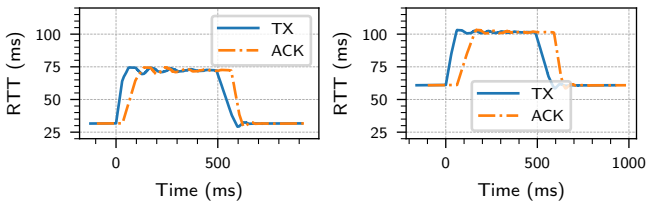

Figure 11: Paced probes reduce oscillation magnitudes. Left shows probe of short RTprop flow, and right shows long.

variations and biases are interesting in their own right, they even occur in packet-level simulation, and to our knowledge have not been documented before.

Multiple hops. Fig. 9 shows that the immediate excess delay created by the probe is more than the transmission delay of the probe. Initially, the sending rate of the probe is higher than the capacity of the hops. As a result, transient packets queue up at the non-bottleneck hop, before also queuing at the bottleneck hop where the other flow is competing. The extra delay subsides when the packets become ACK-clocked in the next RTT.

Different RTprops. Fig. 10 shows oscillations in RTT caused by differences in feedback delay of flows. We provide intuition for the case when the longer RTprop flow probes (the other case is similar). At the start of the probe, the long flow's sending rate is high (increased delay). This reduces the ACK rate (and sending rate) of the short flow. The burst packets take a while to trigger new transmissions (in response to their

ACKs) as the probing flow's RTprop is long (decreased delay). Each time the burst gets paced by the bottleneck and reduces in rate and eventually dies out.

Design of probe. In lieu of these transient variations, we make two choices (illustrated in Fig. 8): (1) we increase and decrease our `cwnd` over an RTT instead of abruptly (this reduces the magnitude of oscillations (Fig. 11), and (2) we wait for two RTTs before starting to measure excess delay. This eliminates the variations due to multiple hops, but the feedback delay based variations may not fully vanish.

We account any oscillations that do not vanish under (E1). This is okay because these oscillations are bounded independent of our design parameters, i.e., (P1) they largely depend on the differences in RTprops, and (P2) do not asymptotically grow with our probe sizes. As a result, the relative impact of these vanishes as we increase our delay/delay variations. We found it hard to mathematically prove (P1/P2) and empirically validated them (not shown).

We measure excess delay as the minimum RTT over the "packets part of probe" in Fig. 8 minus the minimum RTT in the slot before the probe.

5.4 Duration and number of slots

Duration of slots. Since slots allow us to coordinate probes, all slots, including probe and cruise, need to have the same duration, i.e., 4RTTs (§ 5.3.2). We conservatively set the slot size as the 4× the maximum RTT observed in the slot (for both probe and cruise slots). Note, slot durations may differ across flows as RTTs may differ due to noise or differences in RTprops. Empirically, FRCC works despite this (§ 7). Our formal proofs assume that slot durations are equal (**Assumption 2.**). We experimented with a design that hard-codes a large enough slot duration, but deprecated this as FRCC empirically worked without it.

Number of slots. When probes overlap, flows may underestimate capacity (and N_C) and incorrectly decrease `cwnd`. Underestimation occurs because excess delays add up ($\widehat{\Delta d}$ is larger so $\widehat{C} = E/\widehat{\Delta d}$ is smaller).

More slots decrease the probability of such overlaps at the cost of increasing convergence time. We dynamically set the number of slots to be linear in flow count as $k\widehat{N_T}$, with the intuition that $\widehat{N_T} = N_G$ in steady-state. In transient state, $\widehat{N_T}$ may not equal N_G. To ensure enough slots, we set the minimum number of slots to $K_{\min} = 6$. When there are too many probes for the number of slots, the natural increase in delay will increase $\widehat{N_T}$ and the number of slots. For practical reasons, our implementation also clamps slot count above ($K_{\max} = 20$). C.f. BBR [7] uses 8 slots. We investigate the impact of the upper clamp empirically (Appendix E).

Note that with slot count linear in N_G, while the probability of *some* probes colliding is high due to the birthday paradox, the probability of *a particular flow* colliding is low (§ B.5). Thus, each flow has infinitely many rounds where its probes do not collide, and it makes progress towards fairness. Additionally, flows that collide yield bandwidth to other flows, but each flow is equally likely to collide, so on average, the flow throughputs are still equal (§ 6.4, § 7).

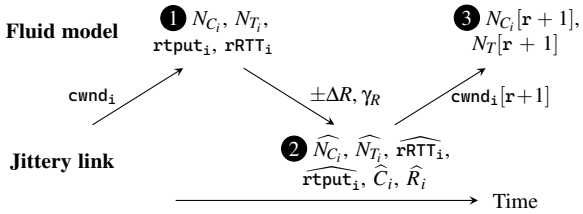

Figure 12: Process for deriving state transition function.

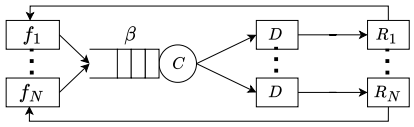

Figure 13: Jittery link (CBR-delay model from [3]). N_G flows, f_1 to f_{N_G}, on a dumbbell topology, with capacity C pkts/sec, infinite buffer, and RTprop of R_i for flow f_i. The jitter boxes can delay packets by D seconds non-deterministically (or arbitrarily) to model (E1/E2) errors in § 5. **Ideal link** uses $D = 0$.

6 Analysis and proofs

We describe highlights of our analyses, with details in Appendix C. We use three network models corresponding to our error models (§ 4): (1) jittery link (E1/E2/E3) (Fig. 13), (2) ideal link (E2/E3), and (3) fluid model (E3) only.

We analyze FRCC under four different settings, each providing unique insights into the FRCC's dynamics and performance guarantees. Our split of these four settings is governed by the tractability of the analyses tools under non-linearity (different RTprops and multiple bottlenecks), stochasticity (probe collisions) and non-determinism (jitter).

§ 6.2 Ideal link, N_G flows, equal RTprop, no probe collisions.
§ 6.3 Jittery link, **2 flows**, equal RTprop, no probe collisions.
§ 6.4 Ideal link, **2 flows**, equal RTprop, **with collisions**.
§ 6.5 Fluid model, different RTProps & multiple bottlenecks.

Based on our analysis of these settings, we conjecture that FRCC guarantees bounded fairness with high probability (over the algorithm's randomness in choosing the probe slot) on well-behaved network topologies (i.e., full rank routing matrix [36]), including those with jitter and multiple hops.

6.1 Approach

Our analyses proceed by deriving a state transition function that describes how the system state evolves after each round of FRCC's `cwnd` updates. We use a three step process to derive this function (Fig. 12):

Notation. We represent state in terms of the current and target flow counts. We use the suffix $[\mathbf{r}+1]$ to denote state in next round, $\langle N_{C_i}[\mathbf{r}+1], N_T[\mathbf{r}+1]\rangle$, and no suffix for current round.

❶ **Fluid reference execution.** We want to make statements about the gap between flow throughputs and error between our estimates and true values. However, due to (E1/E2), even when all `cwnd`s, link capacities, flow count are fixed (non-time varying), there are variations in RTT and throughput. So there is no single true value of RTT, throughput, or link fraction. To deal with this, we define a "fluid reference execution", to define true values. Our proofs describe how these **reference** values (or state) evolve over time. The **real** state, RTT, and throughput hover around the reference values (§ 6.3).

We define the hypothetical fluid reference execution as one that runs parallel to the real one but without (E1/E2). It shares the same network parameters as the real execution (e.g., C, N_G, R_i, etc.). Given some initial $\texttt{cwnd}_i$, we define the reference (true) values for `rRTT`, `rtput`, (and consequently, N_{C_i}, N_T), as the RTT and throughput in the fluid execution. These are solutions to the two fluid model equations: (1) $\texttt{rtput}_i = \texttt{cwnd}_i/\texttt{rRTT}_i$ and (2) $\sum_i \texttt{rtput}_i = C$. These yield a unique solution to $\texttt{rRTT}_i$ and $\texttt{rtput}_i$ [36], and consequently N_{C_i} ($= C/\texttt{rtput}_i$) and N_T ($= (\texttt{rRTT}_i - R_i)/\theta$). Note, we use the same reasoning to define the excess delay created by a probe of size E as $\Delta d = E/C$.

❷ **Bounds on estimates.** We respectively define ΔR and γ_R[2] as additive error in a RTT measurement and multiplicative error in a throughput measurement in the real execution relative to the reference $\texttt{rtput}_i$ and $\texttt{rRTT}_i$. These represent cumulative errors due to (E1/E2). Using these, we derive bounds on FRCC's estimates ($\widehat{\texttt{rRTT}_i}$, $\widehat{\texttt{rtput}_i}$, $\widehat{N_{C_i}}$, $\widehat{N_{T_i}}$) relative to the reference state (similar to Eq. 5.2).

❸ **Next state(s).** We plug-in the bounds into FRCC's `cwnd` update rules (§ 5.2). This yields the (possible) $\texttt{cwnd}_i[\mathbf{r}+1]$ in the next round and consequently the next reference state $\langle N_{C_i}[\mathbf{r}+1], N_T[\mathbf{r}+1]\rangle$, as a function of the current state, design (γ, θ, etc.) and network parameters (ΔR, R_i, C, etc.).

Our analyses study the evolution of this function, assuming non-time varying link capacity and flow count. When these change, we restart the evolution from a new initial state, similar to CCmatic [1]. We show convergence for all initial states.

Assumptions. Since we model FRCC's execution at the granularity of rounds, we make several simplifying assumptions about the slot-level execution. For instance, we assume that rounds and slots of flows align (they have the same durations and start/end times). We assume that flows update `cwnd`s synchronously (i.e., all flows first make estimates, then update their `cwnd`), in reality flows may update `cwnd` one-after-another (sequentially) (we relax this assumption in § 6.4). We detail all these assumptions and more in § C.2.

Apart from describing FRCC's dynamics and performance, the primary purpose of our analysis is to demonstrate that FRCC correctly handles jitter—a worst-case (or adversarial) element controlled by the network. Since, aspects like the randomness we use to decide probing slots which affects probe collisions are not in control of the adversary, we believe it is okay to rely on numerical and empirical methods for validating these components like any other CCA evaluation.

These assumptions make our analysis tractable. Empirically, FRCC works even though these assumptions do not hold (§ 7).

[2]While ΔR is a pure network parameter. γ_R depends on the RTT we maintain. Larger θ decreases γ_R (γ_R is effectively $\frac{RTT}{RTT \pm \Delta R}$, from $\frac{\widehat{\texttt{rtput}_i}}{\texttt{rtput}_i}$).

6.2 Ideal link (Theorem 6.1)

THEOREM 6.1. *On an ideal link (Fig. 13) with equal RTprops, assuming no probe collisions occur, under assumptions in § C.2, FRCC ensures fairness, i.e., every flow's steady-state sending rate is C/N_G, and RTT is $R+\theta N_G$.*

Proof sketch. The proof (Appendix C) mathematically describes for an arbitrary number of flows what we see in Fig. 14 (a) for the behavior of two flows. N_{C_i} and N_T move towards N_G, and in steady-state, $\forall i.\, N_{C_i} = N_T = N_G$. This implies (1) 100% utilization (since $\texttt{delay} = \theta N_T = \theta N_G > 0$) and (2) fairness (equal N_{C_i} imply equal throughputs, since $N_{C_i} = C/\texttt{rtput}_i$ (definition of N_C (§ 6.1)).

Our proof shows that with equal RTprops, there are no (E2) errors and the ideal link is equivalent to the fluid model (Lemma C.1). Thus, our estimates are error-free ($\widehat{N_{C_i}} = N_{C_i}$, $\widehat{N_{T_i}} = N_{T_i}$). We substitute these into the `cwnd` update to get the next state, and state transition function (Eq. C.6, Eq. C.7).

We show that under the transition function, the gap between N_T and N_G, i.e., $|N_T[\mathbf{r}] - N_G|$ decreases multiplicatively (or exponentially fast) each round. Likewise, the gap between $N_{C_i}[\mathbf{r}]$ and N_G, decreases multiplicatively.

For the ideal link, our proof works for all design parameter choices as long as they are non-zero and positive. Since there are no errors in estimates, we do not need clamps on the `cwnd` (i.e., $\delta_H = \infty$, $\delta_L = \infty$). Our probes just need to create non-zero delay ($\gamma > 0$, $\mathcal{D} > 0$), and we need to create non-zero seconds of queueing per flow ($\theta > 0$).

6.3 Jittery link (Theorem 6.2)

THEOREM 6.2. *On a jittery link (Fig. 13), with $N_G = 2$ flows, both having equal RTprops, assuming no probe collisions, and under assumptions in § C.2, FRCC bounds unfairness (avoids starvation). Specifically, it ensures that the ratio of steady-state flow throughputs (in the corresponding fluid reference execution), i.e., `rtput`, is at most $6.6\times$.*

Proof. The evolution of FRCC on jittery link is similar to ideal link. The key difference is the state transition function gives a set of possible next states instead of a unique next state and consequently FRCC converges to region instead of a fixed-point. The lemmas (Lemma C.6, Lemma C.7, Lemma C.8, proved in § C.4), summarized in Fig. 15, show that for all initial states, all trajectories (despite jitter) converge to the shaded region, and then stay in that region. As a result in steady-state, $N_{C_i} \in [6.6, 6.6/5.6]$. This implies that the reference throughputs ($\texttt{rtput}_i$) are at most off by $6.6\times$ (as $N_{C_i} = C/\texttt{rtput}_i$).

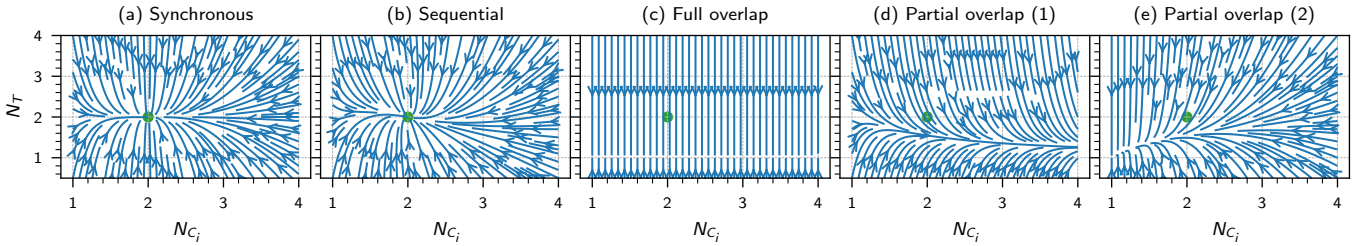

Figure 14: Evolution of $\langle N_{C_i}, N_T \rangle$ over rounds on ideal link for $N_G = 2$ flows. In all the overlaps, N_T moves towards 1. With partial overlap, N_{C_i} increases (link fraction decreases) for the flow that witnesses combined excess delay (d) and decreases for the other flow (e).

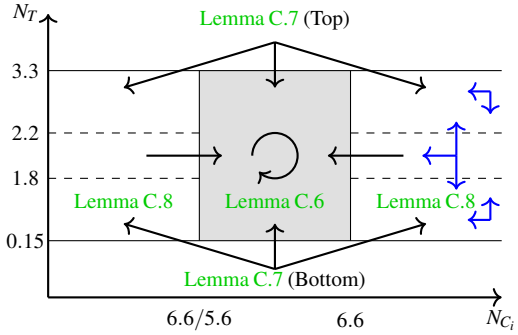

Figure 15: Evolution of $\langle N_{C_i}, N_T \rangle$ over rounds on jittery link.

Since jitter can emulate different "effective" RTprops , the real throughputs can be slightly off this guarantee: if jitter doubles the RTT for one flow, but not the other, its throughput halves, and the throughput ratio could go from 6.6× to 13.2×.

Note, we were only tractably able to write and check the proof of Theorem 6.2 for $N_G = 2$ flows since the number of state variables, and hence the solving time for Z3, grows with N_G.

Parameters. There is a one-to-one relation between the design/network parameters and the performance bounds (e.g., 6.6× in the theorem). Given design/network parameters, we find the performance bounds using binary search (§ C.4). We iterated this a few times to find a reasonable combination of parameters and performance bounds. We set the design parameters as: $\gamma = 4$, $\alpha = 1$, $\delta_H = 1.3$, $\delta_L = 1.25$. In terms of the amount of jitter, our design works as long as: delay is larger than jitter ($\theta > \Delta R/2$), delay variation is larger than jitter ($\mathcal{D} > \Delta R$), and RTT is large enough to bound throughput variations due to RTT variations ($\gamma_R < 2$).

Additionally, we also need $\theta * N_G > R$ for stability, as we use the alternate `cwnd` update (§ 5.2), i.e., less stable update, for jittery link proof. This condition is also needed for ideal link if we use the alternate update.

6.4 Impact of probe collisions (overlaps)

Types of probe collisions. When the probes of two flows collide, either one, neither, or both of them may underestimate capacity, depending on exactly how their probes overlap (Fig. 16). A probe underestimates capacity when it measures the combined excess delay from multiple probes. Since we take the minimum of RTT samples in the probe's measurement interval (T_P), we measure the combined delay only if *the entirety* of T_P overlaps with another flow's probe slot.

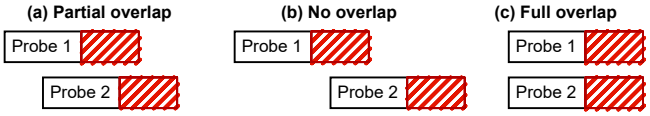

Figure 16: Probe overlap scenarios. Rectangles represent probe slots during which `cwnd` is high. The red hatched portions show intervals where flows measure excess delay. (a) Flow 1 measures combined delay while flow 2 does not. (b) Neither flow measures combined delay. (c) Both flows measure combined delay.

Transition function(s). We update the ideal link transition function to include the impact of probe collisions by updating $\widehat{\Delta d}$. Specifically, if two flows have probe sizes E_1 and E_2, the total excess delay is $(E_1+E_2)/C$. The flows could see either $\widehat{\Delta d_i} = E_i/C$, or the combined delay. We substitute both these possibilities for both the flows, to get different transition functions.

State trajectories. Fig. 14 shows the state trajectories of the system under different alignment of probe slots. We get 5 different functions for the possible probe slot alignments. The first two cases are when the probes do not collide. Two flows may measure the same state and update their `cwnd`s together (synchronous), or one flow may first update the `cwnd`, causing the second flow's round RTT and throughput estimates ($\widehat{\texttt{rRTT}}$ and $\widehat{\texttt{rtput}}$) to change before its `cwnd` update (sequential).[3] In both cases, the system state converges to the fair and efficient fixed-point ($N_T = N_{C_i} = N_G = 2$), shown by the green circle.

With probe collisions, both flows may measure the combined excess delay (full overlap), or only one of them may measure the combined excess delay (partial overlap). In full overlap, both flows update their `cwnd` by the same ratio, and this does not change their link fractions (no motion along the N_{C_i} axis). With partial overlap, the flow that underestimates capacity yields bandwidth to the other flow. Fig. 14 (d) shows the perspective of the flow that yields bandwidth and (e) shows the flow that receives bandwidth.

Note, for two flows, the state of one flow is enough to represent the state of the system because $N_{C_2} = \frac{N_{C_1}}{N_{C_1}-1}$. (d) and (e) are images of each other under this transformation. This is because link fractions sum to 1, $\sum_i 1/N_{C_i} = 1$. Since the fixed point in (e) is $\langle 1,1 \rangle$, the fixed point in (d) is $\langle \infty,1 \rangle$.

[3] If the first flow increases its `cwnd`, the updates remain synchronous because the second flow's $\widehat{\texttt{rtput}}$ and $\widehat{\texttt{rRTT}}$ do not change. The increased `cwnd` increases RTT and decreases throughput of the second flow, these measurements get filtered out by the max/min filters.

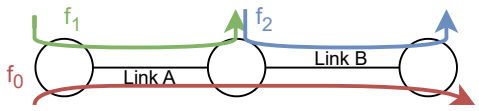

Figure 17: Parking lot topology. Flow f_0 gets observations from both hops, f_1, f_2 only see a single hop.

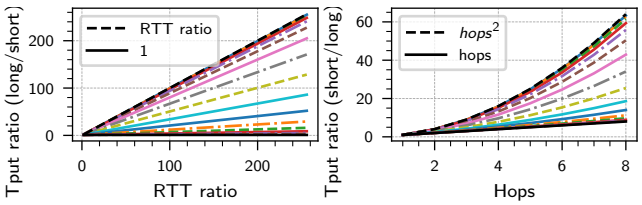

Figure 18: Throughput ratios at the fixed point with different RTprops (**Left**), and parking lot topology (**Right**). The different lines correspond to different choices of probe size ($\gamma\mathcal{D}$) relative to the RTprop of the short flow (with $\gamma\mathcal{D}/R$ ranging from 2^{-7} to 2^{8}). Large probe size yields better (lower) throughput ratios.

FRCC is fair despite collisions. The phase portraits show system evolution if the same transition function is taken in every round. In reality, the system state will evolve under a superposition of these transition functions each taken with some probability. (a), (b) move the system towards fairness and (c) does not change fairness (no change to link fractions). (d) and (e) do cause one flow to yield bandwidth to another, but the system is equally likely to take any of these transitions. This is because, both flows have equal slot and measurement durations, and we resample probing slot in each round. Hence they are equally likely to measure combined excess delay. This explains why FRCC is fair despite probe collisions as the effects of (d, e) cancel out and (a, b, c) do not cause unfairness.

Note, collisions do cause N_T to decrease creating instability and unfairness at short time scales (Appendix E).

6.5 Fluid model analysis (different RTprops and multiple bottlenecks)

With different RTprops or multiple bottlenecks, the excess delay (Δd) is no longer equal to the transmission delay of the probe, i.e., $\Delta d \neq E/C$. This creates a bias in our capacity estimate, and consequently rate allocation. Instead of equal throughputs, FRCC's throughput ratio may be as bad as the RTT ratio (not RTprop ratio). With parking lot topology, instead of proportional fairness [26] (i.e., a throughput ratio of `hops`), FRCC's throughput ratio may be as large as $\texttt{hops}^2$.

Different RTprops. When the short RTT flow probes, the excess delay is more than expected. Conversely, when the long flow probes, the excess delay is less than expected. We give intuition for the short flow probe. During the probe, short flow's rate increases and long's decreases by the same amount. However, since the long flow has larger RTprop, the packets in its pipe reduce by a larger amount. These go into the queue, increasing the delay beyond the probe's transmission delay.

We compute the exact value of the biased excess delay using the fluid model (Appendix D). This yields bias in $\widehat{C}$, and consequently bias in $\widehat{N_C}$. We include these biases in the transition function and solve for the fixed-point (i.e., next state = current state). We use numerical methods to approximate the fixed-point as we do not get closed-form expressions for the transition function in this analysis (Appendix D). Fig. 18 shows the throughput ratios at the fixed-point. The ratio depends on the size of our probes (i.e., $\gamma\mathcal{D}$) relative to the short flow's RTprop. With large probes, the throughput ratio approaches 1, while with small probes, it approaches the RTT ratio between flows.

Multi-bottleneck parking lot topology. Fig. 17 illustrates the parking lot topology where different flows witness congestion signals from different number of hops. As a result, if f_1 estimates the queuing delay to be `delay`, then f_0 witnesses a delay of $\texttt{hops}\cdot\texttt{delay}$. Since CCAs coordinate fairness through congestion signals (§ 2), this creates bias in *all* end-to-end CCAs, shifting the rate allocation away from max-min fairness [2, 26, 34] (equal rates for all flows). For instance, Copa/Vegas/FAST achieve proportional fairness [26, 36], where flows using more bottleneck resources receive proportionally less bandwidth. While FRCC would achieve proportional fairness with just the contract bias, the bias in FRCC's capacity estimate results in throughput ratios larger than proportional fairness.

Similar to the different RTprop case, we compute this bias and FRCC's fixed-point (Appendix D). Fig. 18 shows the resulting throughput ratios, with FRCC's throughput ratio falling between `hops` and $\texttt{hops}^2$ depending on the probe size.

7 Empirical evaluation

We evaluate FRCC with four goals in mind: (1) illustrate the performance properties of FRCC, while (2) validating our proofs, (3) measure performance relative to existing CCAs, and (4) explore when FRCC breaks.

Implementation & Methodology. We implement FRCC in the Linux kernel as a pluggable kernel module. We set FRCC's design parameters as guided by our jittery link proof (§ 6.3), we set: $\gamma = 4$, $\alpha = 1$, $\delta_H = 1.3$, $\delta_L = 1.25$, $\mathcal{D} = \theta = 10$ ms, $T_R = 30$ seconds, and T_P = 1RTT. Our whole goal was to scale to arbitrarily large link capacities. Our proof shows that these parameter choices work for all link capacities and RTprops, as long as the buffers are large enough (§ 2, § 6.3). Note, even though we set $\mathcal{D} = \theta = 10$ ms, this is for worst-case jitter, our design handles much more than 10 ms of jitter in experiments.

We compare against Cubic [16], Reno [18], BBRv1 [7] (Linux kernel v5.15.0), BBRv3 [13], and Copa [5, 33]. We use mahimahi [28] and mininet [9] to emulate and test a variety of network scenarios. We perform various network parameter sweeps. By default, we use a dumbbell topology with $C = 48$ Mbps, $R = 50$ ms, $N_G = 3$ flows, and buffer size $= \infty$ (we set buffer $= 3\texttt{BDP}$ for Reno/Cubic, otherwise they set $\texttt{cwnd} = \infty$).

We use `tcpdump` at the senders to measure the throughput and RTT. We study the link utilization, flow throughput, fairness (Jain's fairness index (JFI) [19], and throughput ratio

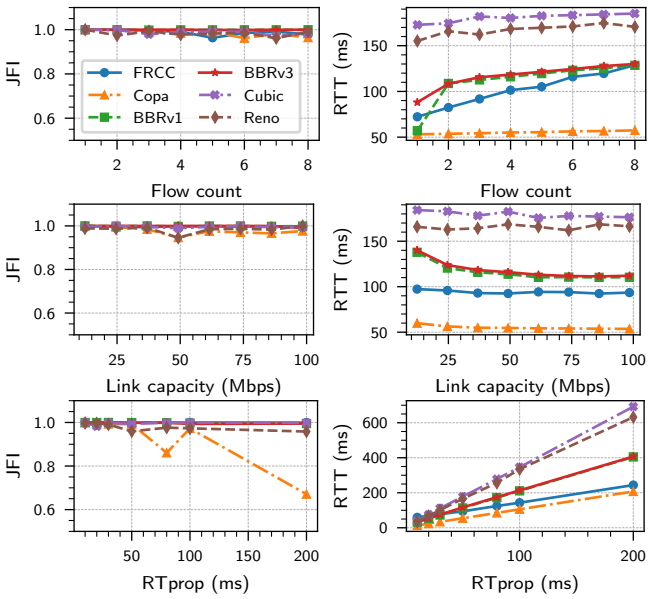

Figure 19: Parameter sweeps with ideal link. We omit RTT for Cubic, it bloats buffers building very high delays and skews the plot.

$\left(\max_{i,j} \frac{\texttt{throughput}_i}{\texttt{throughput}_j}\right)$), RTTs (latency), and convergence time. We run experiments for 5 mins, and measure these metrics in steady-state (i.e., minute 3 onward).

Parameter sweeps without jitter. We sweep flow count (1 to 8 flows), link capacity (12 to 96 Mbps), and RTprop (10 to 200 ms). This is a wide sweep that is at the limits of what mahimahi emulation can support. Fig. 19 shows that across all sweeps—flow count, capacity, and RTprop—all CCAs achieve fairness (JFI $\approx$ 1), except Copa, which struggles at high RTprops. All flows maintain non-zero queueing in steady-state yielding a link utilization of roughly 100% (not shown).

The RTT plots agree with the expected RTTs of the CCAs. With increasing flow count, FRCC's RTT increases linearly by $\theta = 10$ ms per flow (RTT $= R + \theta N_G$). BBR's RTT also increases linearly. It operates in `cwnd`-limited mode (when $N_G > 1$) and maintains $\alpha_{\texttt{BBR}}$ packets (not seconds) per flow ($\texttt{cwnd}_i = 2\widehat{C}_i\widehat{R}_i + \alpha_{\texttt{BBR}}$, RTT $= 2R + N_G\alpha_{\texttt{BBR}}/C$ [3]). Copa's RTT also increases linearly maintaining $1/\delta_{\texttt{Copa}}$ packets of queueing per flow (RTT $= R + N_G/(\delta_{\texttt{Copa}}C)$ [3, 5]). Reno/Cubic fill up buffers and have an RTT of $4R$ independent of the flow count (RTT $= R + \texttt{buffer}/C$, $\texttt{buffer} = 3\texttt{BDP}$)

Looking at the capacity sweep, FRCC's RTT remains constant regardless of link capacity. Similarly, Cubic and Reno maintain constant RTT of $4R$. In contrast, BBR's RTT decreases with increasing capacity as the seconds of queueing (transmission delay) of its $\alpha_{\texttt{BBR}}$ packets in queue diminishes. Copa follows the same pattern, with its RTT decreasing as capacity increases. This vanishing queueing delay (or delay variation) with increasing link capacity is precisely what leads to starvation under jitter. FRCC avoids this pitfall by maintaining delay and delay variation that remain larger than network jitter regardless of link capacity.

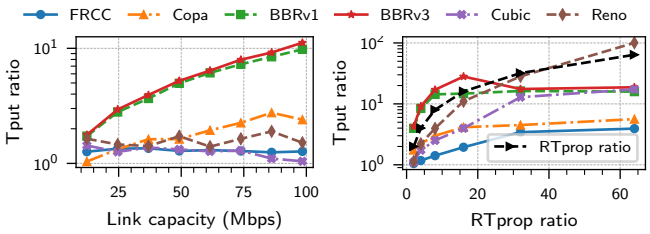

Figure 20: Jitter (different RTprops). [**Left**] Three flows with RTprop of 10, 20, and 30 ms, with varying link capacity. [**Right**] Two flows with varying RTprop ratio. FRCC's throughput ratio is between 1 and RTprop ratio even when RTprops are off by 60×.

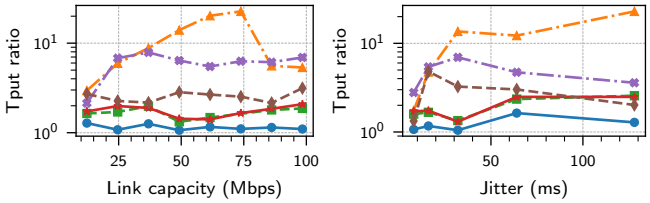

Figure 21: Jitter (ACK aggregation). Three flows with $R = 32$ ms. Flow f_0's path has 32 ms of ACK aggregation. [**Left**] Varying link capacity. [**Right**] Varying amount of ACK aggregation on flow f_0's path.

Finally, the RTT vs RTprop plot is also consistent with the expected RTTs. The slope of RTT vs RTprop is 4 for Reno/Cubic, 2 for BBR, and 1 for FRCC and Copa.

The absolute RTTs (and RTT ranking across CCAs) varies depending on the design parameters, flow count, link capacity (transmission delay), RTprop, and buffer size. Trends in RTTs matter more than the absolute values.

Sweeps with jitter. We study two manifestations of jitter: (1) different RTprops and (2) ACK aggregation. These are settings where FRCC shines. Fig. 20 [Left] shows flows with slightly different RTprops (10, 20, and 30 ms), FRCC/Cubic/Reno yield close to equal throughputs (ratio $\approx$ 1). BBR causes unfairness which increases with link capacity and ends up starving the shorter RTprop flow (e.g., Fig. 1). Copa also causes increasing unfairness with increasing link capacity.

To emulate ACK aggregation, we implement an aggregator parameterized with a burst size in milliseconds and a burst rate. The aggregator collects all ACKs in the burst size period and sends them at the burst rate. We set the burst rate to ensure that the aggregator does not become the bottleneck. We sweep the link capacity in Fig. 21 [Left] and burst size in Fig. 21 [Right].

Copa and Cubic/Reno cause unfairness which increases with link capacity, eventually starving flows. Copa starves the flow with aggregation while Reno/Cubic starve the other flows (Fig. 2). FRCC maintains roughly equal throughputs for flows even when the jitter is 128 ms—10× what FRCC was tuned to tolerate. BBR exhibits slight unfairness.

Vastly different RTprops and multiple bottlenecks. Fig. 20 [Right] shows throughput ratios of two flows with varying RTprop ratio. The throughput ratios for all CCAs increase with RTprop ratio. FRCC's throughput ratio is lower (more fair) than other CCAs even when the RTprops are off by 64×. FRCC's

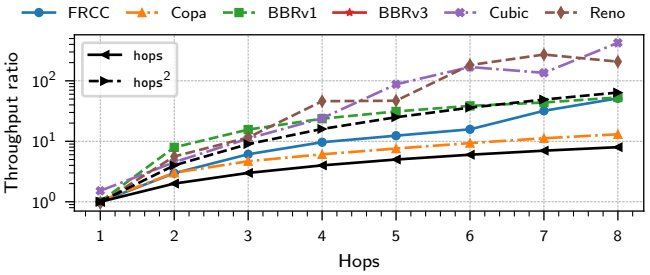

Figure 22: Parking lot topology. FRCC's empirical throughput ratio is between hops and hops^2. We do not show BBRv3 as we have separate VMs for BBRv3 and mininet (for emulating parking lot).

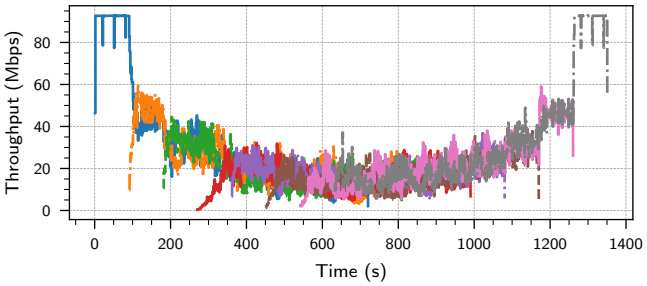

Figure 23: FRCC converges to fair rates as flows enter and leave. Each line shows the throughput of a different flow.

throughput ratio lies between 1 and RTT ratio and is consistent with our analysis (§ 6.5). Note, RTprop ratio > RTT ratio.

Fig. 22 shows results with the parking lot topology with $C =$ 100 Mbps and RTprop = 10 ms for all flows. FRCC's throughput ratio lies between hops and hops^2, consistent with our analysis (§ 6.5). The throughput ratio for Copa is $\approx$ hops (proportional fairness), while that for BBR/Cubic is worse than hops^2.

Convergence time. Fig. 23 shows 8 flows competing on a 96 Mbps link with 50 ms RTprop. Each flow enters 90 seconds after the previous one and runs for $8 \times 90 = 720$ seconds. Practically, FRCC's convergence time is slow and takes tens of seconds to converge. FRCC takes $O(\log(\texttt{BDP}))$ rounds to converge. When there are four flows, a slot is 4 RTTs or 360 ms (RTT $= R + \theta N_G = 50 + 4 * 10$), and a round is $kN_G = 8$ slots or 3 seconds. C.f. a round for BBR is $8 * R = 0.4$ seconds.

8 Discussion

Summary. We presented FRCC, the first CCA that provably achieves fairness despite network jitter, using our key insight of of decoupling fair rate coordination. Our formal analysis demonstrates that FRCC converges exponentially fast to efficient and fair rate allocations with bounded throughput ratios even under worst-case jitter patterns. Our empirical evaluation validates these theoretical guarantees, showing that FRCC maintains fairness in scenarios where existing CCAs starve.

Limitations and future work. While FRCC represents significant progress, three key limitations may need to be addressed for a practical deployment: (L1) operation with shallow buffers, (L2) coexistence with other CCAs, and (L3) convergence time. Of these, we believe the convergence time is the most critical. For instance, even widely deployed CCAs like Reno, Cubic, and BBR struggle with shallow buffers [8] and coexisting with each other [35].

We outline three promising directions for addressing these limitations. First, to handle shallow buffers, a contract based on packet loss [12, 25] rather than delay could be explored. However, directly adopting Reno's loss-rate encoding is insufficient, as its sawtooth behavior causes under-utilization when buffers are smaller than the BDP. Second, our current contract encodes information in the *magnitude* of delay. Future work could explore encoding the fair rate in the *frequency* or *pattern* of delay variations. A prior proposal for AIMD on delay [3] follows this direction, but we found this specific design can lead to starvation on multi-bottleneck paths even without jitter [2].

Third, while FRCC's asymptotic convergence time of $\log(N_G\texttt{BDP})$ RTTs is competitive, its practical convergence is longer due to (1) a long, multi-RTT probing slot, and (2) $O(N_G)$ slots in a round. Further research on capacity estimation could reduce the probing duration. Alternatively, an initial deployment could trade theoretical robustness in favor of convergence time. For instance, the long probing slot helps deal with transient RTT variations with different RTprops and multiple bottlenecks (Figures 9, 10, and 11). A shorter slot duration and fewer slots per round would help improve convergence at the cost of modest unfairness in those specific cases, while still solving the core starvation problem in Figures 1 and 2.

Takeaways. Our work may influence the CCA landscape in at least four ways. First, FRCC could be deployed with a pragmatic tuning (as described above) or after further research addresses its limitations. Second, parts of FRCC (e.g., link fraction coordination, link capacity estimation, transient/persistent RTT variations) may inform other CCA designs. Third, further research may conclude that fairness is a much harder problem where end-to-end methods like FRCC come at an unbearable cost, bolstering the case for deployment of in-network solutions or informed over-provisioning of the Internet.

Finally, our design would not have been possible without formal reasoning and worst-case modeling of the network. We began by stating assumptions about the network and desired performance guarantees, then worked backwards to derive a CCA that meets these requirements. This (1) revealed high-level key insights and (2) guided our low-level design decisions. We believe this is a promising methodology in general for developing robust network control algorithms with predictable performance in challenging environments.

Acknowledgments

We are grateful to the anonymous reviewers and our shepherd, Srikanth Sundaresan, for their comments that helped improve our paper. This work was supported in part by NSF grants CNS-2403026, CNS-2212102, and CNS-2212390. We dedicate this paper to the memory of our beloved friend, colleague, and mentor, Prateesh Goyal.

References

[1] Anup Agarwal, Venkat Arun, Devdeep Ray, Ruben Martins, and Srinivasan Seshan. "Towards provably performant congestion control". In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 951–978. URL: https://www.usenix.org/conference/nsdi24/presentation/agarwal-anup.

[2] Anup Agarwal, Venkat Arun, and Srinivasan Seshan. *Contracts: A unified lens on congestion control robustness, fairness, congestion, and generality*. 2025. arXiv: 2504.18786 [cs.NI]. URL: https://arxiv.org/abs/2504.18786.

[3] Venkat Arun, Mohammad Alizadeh, and Hari Balakrishnan. "Starvation in End-to-End Congestion Control". In: *Proceedings of the 2022 ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022.

[4] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. "Toward Formally Verifying Congestion Control Behavior". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 1–16. URL: https://doi.org/10.1145/3452296.3472912.

[5] Venkat Arun and Hari Balakrishnan. "Copa: Practical Delay-Based Congestion Control for the Internet". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 329–342. URL: https://www.usenix.org/conference/nsdi18/presentation/arun.

[6] L.S. Brakmo and L.L. Peterson. "TCP Vegas: end to end congestion avoidance on a global Internet". In: *IEEE Journal on Selected Areas in Communications* 13.8 (1995), pp. 1465–1480.

[7] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. "BBR: Congestion-Based Congestion Control". In: *ACM Queue* 14, September-October (2016), pp. 20–53. URL: http://queue.acm.org/detail.cfm?id=3022184.

[8] Neal Cardwell, Ian Swett, and Joseph Beshay. *BBR Congestion Control*. Internet-Draft draft-ietf-ccwg-bbr-01. Work in Progress. Internet Engineering Task Force, Oct. 2024. 79 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-ccwg-bbr/01/.

[9] Mininet Project Contributors. *Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet*. [Online; accessed 2025-04-17]. URL: https://mininet.org/.

[10] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. "PCC: Re-architecting congestion control for consistent high performance". In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 2015, pp. 395–408.

[11] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. "What do packet dispersion techniques measure?" In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*. Vol. 2. IEEE. 2001, pp. 905–914.

[12] Sally Floyd, Mark Handley, Jitendra Padhye, and J¨ org Widmer. "Equation-based congestion control for unicast applications". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '00. Stockholm, Sweden: Association for Computing Machinery, 2000, pp. 43–56. URL: https://doi.org/10.1145/347059.347397.

[13] *google/bbr at v3*. [Online; accessed 2025-04-17]. URL: https://github.com/google/bbr/tree/v3.

[14] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. "ABC: A Simple Explicit Congestion Controller for Wireless Networks". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 353–372. URL: https://www.usenix.org/conference/nsdi20/presentation/goyal.

[15] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. "Elasticity detection: a building block for internet congestion control". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 158–176. URL: https://doi.org/10.1145/3544216.3544221.

[16] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-Friendly High-Speed TCP Variant". In: *SIGOPS Oper. Syst. Rev.* 42.5 (July 2008), pp. 64–74. URL: https://doi.org/10.1145/1400097.1400105.

[17] Feixue Han, Qing Li, Peng Zhang, Gareth Tyson, Yong Jiang, Mingwei Xu, Yulong Lan, and ZhiCheng Li. "ETC: An Elastic Transmission Control Using End-to-End Available Bandwidth Perception". In: *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. Santa Clara, CA: USENIX Association, July 2024, pp. 265–284. URL: https://www.usenix.org/conference/atc24/presentation/han.

[18] Janey C. Hoe. "Improving the Start-up Behavior of a Congestion Control Scheme for TCP". In: *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '96. Palo Alto, California, USA: Association for Computing Machinery, 1996, pp. 270–280. URL: https://doi.org/10.1145/248156.248180.

[19] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. "A quantitative measure of fairness and discrimination". In: *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* 21.1 (1984).

[20] Dina Katabi, Mark Handley, and Charlie Rohrs. "Congestion Control for High Bandwidth-Delay Product Networks". In: *SIGCOMM Comput. Commun. Rev.* 32.4 (Aug. 2002), pp. 89–102. URL: https://doi.org/10.1145/964725.633035.

[21] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. "Swift: Delay is Simple and Effective for Congestion Control in the Datacenter". In: *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 514–528. URL: https://doi.org/10.1145/3387514.3406591.

[22] K. Lai and M. Baker. "Measuring bandwidth". In: *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*. Vol. 1. 1999, 235–245 vol.1.

[23] Kevin Lai and Mary Baker. "Measuring link bandwidths using a deterministic model of packet delay". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '00. Stockholm, Sweden: Association for Computing Machinery, 2000, pp. 283–294. URL: https://doi.org/10.1145/347059.347557.

[24] Steven H. Low, Larry L. Peterson, and Limin Wang. "Understanding TCP Vegas: a duality model". In: *J. ACM* 49.2 (Mar. 2002), pp. 207–235. URL: https://doi.org/10.1145/506147.506152.

[25] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm". In: *SIGCOMM Comput. Commun. Rev.* 27.3 (July 1997), pp. 67–82. URL: https://doi.org/10.1145/263932.264023.

[26] J. Mo and J. Walrand. "Fair end-to-end window-based congestion control". In: *IEEE/ACM Transactions on Networking* 8.5 (2000), pp. 556–567.

[27] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[28] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. "Mahimahi: Accurate Record-and-Replay for HTTP". In: *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, July 2015, pp. 417–429. URL: https://www.usenix.org/conference/atc15/technical-session/presentation/netravali.

[29] Adithya Abraham Philip, Rukshani Athapathu, Ranysha Ware, Fabian Francis Mkocheko, Alexis Schlomer, Mengrou Shou, Zili Meng, Srinivasan Seshan, and Justine Sherry. "Prudentia: Findings of an Internet Fairness Watchdog". In: *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM SIGCOMM '24. Sydney, NSW, Australia: Association for Computing Machinery, 2024, pp. 506–520. URL: https://doi.org/10.1145/3651890.3672229.

[30] Adithya Abraham Philip, Ranysha Ware, Rukshani Athapathu, Justine Sherry, and Vyas Sekar. "Revisiting TCP Congestion Control Throughput Models & Fairness Properties at Scale". In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC '21. Virtual Event: Association for Computing Machinery, 2021, pp. 96–103. URL: https://doi.org/10.1145/3487552.3487834.

[31] Rayadurgam Srikant and Tamer Başar. *The mathematics of Internet congestion control*. Springer, 2004.

[32] *SymPy*. [Online; accessed 2025-04-25]. URL: https://www.sympy.org/en/index.html.

[33] *venkatarun95/genericCC*. [Online; accessed 2025-04-17]. URL: https://github.com/venkatarun95/genericCC.

[34] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. "Poseidon: Efficient, Robust, and Practical Datacenter CC via Deployable INT". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 255–274. URL: https://www.usenix.org/conference/nsdi23/presentation/wang-weitao.

[35] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. "Modeling BBR's Interac-

tions with Loss-Based Congestion Control". In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 137–143. URL: https://doi.org/10.1145/3355369.3355604.

[36] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. "FAST TCP: Motivation, Architecture, Algorithms, Performance". In: *IEEE/ACM Transactions on Networking* 14.6 (2006), pp. 1246–1259.

[37] Keith Winstein and Hari Balakrishnan. "TCP Ex Machina: Computer-Generated Congestion Control". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 123–134. URL: https://doi.org/10.1145/2486001.2486020.

A Artifact Appendix

Abstract. We release (1) FRCC's Linux kernel module implementation, (2) code for theoretical analysis and proofs, and (3) scripts to run and plot empirical experiments.

Scope. We hope our artifact allows others to verify our theoretical and empirical claims as well as build upon our work.

For the theoretical analysis and proofs, we include code to reproduce Figures 14, 15, 18 and Theorems 6.1, 6.2.

For the empirical experiments, we include code to reproduce Figures 1, 2, 19, 20, 21, 23. We do not include code to reproduce experiments with BBRv3 and the parking lot topology (i.e., Figure 22) as we have separate VMs for these experiments.

Contents. Our artifact is organized across three repositories:

(1) https://github.com/108anup/frcc/tree/main. This is the main repository that includes the other two repositories as submodules. It contains documentation and the code for reproducing FRCC's theoretical analysis and proofs.

(2) https://github.com/108anup/frcc_kernel/tree/main. This contains FRCC's Linux kernel module implementation.

(3) https://github.com/108anup/cc_bench/tree/main. This contains the scripts for running and plotting empirical experiments.

The main repository (https://github.com/108anup/frcc/tree/main) contains a detailed `README.md` documenting dependencies, setup, known issues, and commands for reproducing all the experiments.

Hosting. At the time of submission, the main repository (https://github.com/108anup/frcc/tree/main) is at commit `4cedada281fb2d7df04f6dad476a30e44be6fd19` on the `main` branch. This commit includes the kernel and benchmarking repositories as submodules.

Requirements. The artifact does not require specialized hardware. Since our artifact modifies the kernel through the kernel module, we recommend running experiments in a virtual machine or on a dedicated host. All evaluations in the paper were performed on an x86_64 server-class machine with 64 cores and 256 GB RAM, allowing multiple concurrent experiments. Users with fewer resources can scale down the degree of concurrency (see `README.md` for details).

B Design details

B.1 FRCC Pseudocode

For simplicity we do not show (1) RTprop probes (§ B.4), and (2) pacing the probe's gain and drain over an RTT (instead the pseudocode just shows abrupt changes in `cwnd`) (§ 5.3), and (3) pacing rate.

Algorithm 2: FRCC pseudocode.

```
Function OnACK(Seq ack, RTT rtt)
    updateEstimates(ack, rtt)
    if probing & shouldInitProbeEnd() then ▷ See § 5
        cwnd ← prev_cwnd ▷ Drain probe's excess
    end
    if slotEnded(ack, rtt) then ▷ See § 5
        if probing then
            probing ← False
            updateCwnd() ▷ See Alg. 1
        end
        if shouldProbe() then ▷ See § 5
            startProbe()
        end
        startSlot() ▷ Reset sRTT (if not probing)
    end
    if roundEnded() then ▷ kN_max slots elapsed
        ▷ Reset rRTT, rtput
    end
end
Function updateEstimates(Seq ack, RTT rtt)
    R̂ ← min(R̂, rtt)
    rRTT ← min(rRTT, rtt)
    if probing & partOfProbe(ack) then ▷ See § 5
        Δd̂ = min(Δd̂, rtt − sRTT)
    else if not probing then
        sRTT ← min(sRTT, rtt)
        rtput ← max(rtput, cwnd/rtt)
    end
end
Function startProbe()
    probing ← True
    Δd̂ ← ∞
    prev_cwnd ← cwnd
    N̂_T ← (rRTT − R̂)/θ
    E ← max(γN̂_T rtput D, 1)
    cwnd ← cwnd + E
end
```

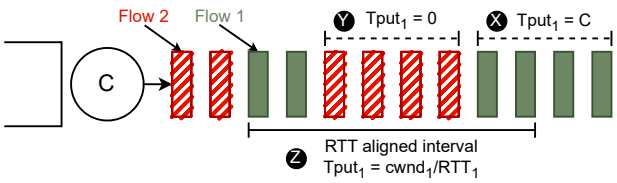

Figure 24: Measuring $\widehat{\texttt{rtput}}$ over non-RTT-aligned intervals can yield a measurement of 0 or C. Measuring over RTT-aligned intervals better matches throughput.

B.2 Probe size derivation

Under our probe size ($E = \gamma\widehat{N_T}\widehat{\texttt{rtput}}\mathcal{D}$), we derive the impact of ΔR noise in $\widehat{\Delta d}$ on our $\widehat{C}$ estimate and consequently its impact on deciphering which of N_C or $\widehat{N_T}$ is bigger.

We used this derivation in inverse (from bottom to top) to derive the probe size in the first place.

$$\widehat{C} = \frac{E}{\widehat{\Delta d}} = \frac{E}{\Delta d \pm \Delta R} = \frac{CE}{E \pm C\Delta R}$$

We get the last equality by substituting $\Delta d = E/C$. Substituting our probe size,

$$\widehat{C} = \frac{C\gamma\widehat{N_T}\widehat{\texttt{rtput}}\mathcal{D}}{\gamma\widehat{N_T}\widehat{\texttt{rtput}}\mathcal{D} \pm C\Delta R}$$

Substituting $\widehat{C}$ into $\widehat{N_C}$ computation,

$$\widehat{N_C} = \frac{\widehat{C}}{\widehat{\texttt{rtput}}} = \frac{C\gamma\widehat{N_T}\mathcal{D}}{\gamma\widehat{N_T}\widehat{\texttt{rtput}}\mathcal{D} \pm C\Delta R}$$

Substituting $\widehat{N_C}$ in,

$$\frac{\widehat{N_T}}{\widehat{N_C}} = \frac{\gamma\widehat{N_T}\widehat{\texttt{rtput}}\mathcal{D} \pm C\Delta R}{C\gamma\mathcal{D}} = \frac{\widehat{N_T}\widehat{\texttt{rtput}}}{C} \pm \frac{\Delta R}{\gamma\mathcal{D}}$$

Substituting $C = N_C \cdot \texttt{rtput}$ (definition of N_C (Appendix D)),

$$\frac{\widehat{N_T}}{\widehat{N_C}} = \frac{\widehat{\texttt{rtput}}}{\texttt{rtput}}\frac{\widehat{N_T}}{N_C} \pm \frac{\Delta R}{\gamma\mathcal{D}}$$

Note, we define the "true" quantities Δd, N_C, $\texttt{rtput}$ based on the fluid model and define ΔR as error in observations in jittery link relative to fluid model (§ 6.1, § C.1). This is why it is okay to use the substitutions $\Delta d = E/C$ and $C = N_C \cdot \texttt{rtput}$.

B.3 Measuring throughput over packet-timed RTTs

We argue that for `cwnd`-based CCAs (1) $\widehat{\texttt{rtput}}$ should be measured over intervals aligned with packet-timed RTTs (i.e., the interval starts with sending a packet and ends with receiving its ACK), and (2) that measuring over multiple aligned intervals (to get longer intervals) provides no additional benefit.

As shown in Fig. 24, the throughput measured over non-RTT-aligned intervals can be anywhere between 0 and C (the link capacity). This occurs due to packet interleaving across flows and exists even on ideal links without jitter. Specifically, in interval X, only packets of flow 1 are served, resulting in throughput equal to C, while in interval Y, only packets of flow 2 are served, yielding zero throughput for flow 1.

In contrast, in the RTT-aligned interval Z, the packets ACKed, between sending a packet and receiving its ACK, are exactly those that were in flight when the packet was sent, which equals the flow's `cwnd`. This gives a throughput estimate of `cwnd`/RTT, where RTT is the round-trip time seen by the last ACK in the interval. While this RTT value may contain noise due to jitter, such noise can persist indefinitely under our network model, causing the same throughput measurement to repeat. Consequently, throughput measurements over multiple RTT-aligned intervals offer no improvement in the worst-case.

Note, this does not contradict [1], which showed that longer measurement intervals yield better estimates, as this is true for rate-based control. Specifically, with rate-based control, packets transmitted (and ACKed) within an RTT may not equal `cwnd` (when the `cwnd` cap is not hit, or if there is no `cwnd`).

Note, pacing may only partially address interleaving. Jitter before the bottleneck can still reorder packets across flows. Further, with `cwnd`-based control, pacing yields no improvement over RTT-aligned estimates anyway.

B.4 RTprop estimate

We can only measure RTprop when queues are drained [7]; however, FRCC maintains θN_G seconds of delay in steady-state. To drain this queue, we launch RTprop probes every $T_R = 30$ seconds, similar to BBR [7]. During these probes, FRCC reduces `cwnd` to 4 packets for one "max RTT", a conservative upper bound on the RTT of competing flows. This ensures that the probe lasts long enough for all flows, with potentially different RTprops, to simultaneously observe the drain.

We set max RTT to "R_{max}+ max slot delay", where max slot delay is the maximum of RTT $-\widehat{R}$ samples in the latest slot. We use $R_{max} = 100$ ms. This is similar to BBR's RTT probe duration of (hard-coded) 200 ms [7].

Note, this only works if the RTprop probes occur together. We assume that senders' clocks are synchronized and launch the probe every 30th second on the clock. This keeps our implementation simple. Today, NTP (network time protocol) makes it relatively easy to get clock synchronization with < 10 millisecond-level accuracy. Note, this accuracy depends on RTprop between NTP servers and senders, and not on the RTprop between senders or between senders and receivers.

Note that we could instead use BBR's mechanism of launching RTT probes when the $\widehat{R}$ expires (i.e., 10 seconds elapsed since $\widehat{R}$ last changed) to implicitly synchronize the RTT probes. In our empirical evaluation, BBR's RTT probe always synchronize regardless of injected jitter, though we do not have a formal proof for this.

B.5 Probability of probe collisions

If there are kN slots and N flows probing. The probability of a particular flow *not* colliding is:

$$\left(\frac{kN-1}{kN}\right)^{N-1} \to e^{-1/k} \text{ as } N \to \infty$$

Hence, the probability of a particular flow colliding is bounded by $1-e^{-1/k}$.

C Proof details

C.1 Preliminaries

Fluid reference execution. Say flow f_i has congestion window $\texttt{cwnd}_\texttt{i}$, round-trip propagation delay R_i. We look at the execution of these choices on a fluid model of the network with link capacity C and infinite buffer.

Under this execution, below we define "reference" state variables: (1) "current flow count" (from the point of view flow i) (N_{C_i}), and (2) "target flow count" (N_T). Recall, the current flow count is inverse of the fraction of link the flow consumes, and target flow count is based on the contract's target delay function, i.e., $N_T = \texttt{Contract}_\texttt{Func}(\texttt{delay}) = \texttt{delay}/\theta$.

$$N_T \text{ solves } \sum_i \frac{\texttt{cwnd}_\texttt{i}}{\texttt{rRTT}_\texttt{i}} = \sum_i \frac{\texttt{cwnd}_\texttt{i}}{N_T\theta + R_i} = C \tag{C.1}$$

$$N_{C_i} = \frac{\sum_j \texttt{rtput}_\texttt{j}}{\texttt{rtput}_\texttt{i}} = \frac{C}{\texttt{rtput}_\texttt{i}}, \text{ with, } \texttt{rtput}_\texttt{j} = \frac{\texttt{cwnd}_\texttt{j}}{\texttt{rRTT}_\texttt{j}} \tag{C.2}$$

$$\text{For } R_i = R, N_T = \frac{\sum_i \texttt{cwnd}_\texttt{i} - CR}{C\theta} \tag{C.3}$$

We also define "$\texttt{rRTT}_\texttt{i}$" ($= N_T\theta + R_i$) and "$\texttt{rtput}_\texttt{i}$" ($= \texttt{cwnd}_\texttt{i}/\texttt{rRTT}_\texttt{i}$) as the "reference" RTT and throughput values.

Summary of notation. We track four variables: $\texttt{rRTT}_\texttt{i}$, $\texttt{rtput}_\texttt{i}$, N_{C_i}, N_T (Table 2).

For each of these variables there are: (1) values in the reference execution, and (2) estimates maintained by the FRCC. Further, all these quantities change as rounds progress. We use the following notation to differentiate between these uses.

- Symbols without any annotation or with $[\texttt{r}]$ suffix (e.g., $\texttt{rRTT}_\texttt{i}$ or $\texttt{rRTT}_\texttt{i}[\texttt{r}]$) denote the state variables in round $\texttt{r}$ of the reference execution. We also use symbols without $[\texttt{r}]$ suffix to refer to state after a probe (in addition to at boundaries of rounds).
- Symbols with hats (e.g., $\widehat{\texttt{rRTT}_\texttt{i}}[\texttt{r}]$) denote the estimates or state variables maintained by FRCC.
- As a shorthand we drop the $[\texttt{r}]$ suffix when clear from context, e.g., we use $\widehat{N_{C_i}}[\texttt{r}_\texttt{F}] = \widehat{N_{C_i}}[\texttt{r}+1]$ and $\widehat{N_{C_i}}$ to denote state in the next round and current round respectively.

C.2 Assumptions

Since our analyses are at the level of rounds, we make several assumptions about the slot-level behavior.

> "The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise." — Edsger Dijkstra

Assumption 1. RTprop probe works. We assume that our RTprop probing mechanism works; i.e., flows can simultaneously drain queues and get an RTT measurement without queueing delays. This may still have errors due to (E1/E2). Note, there is no error due to capacity probes (E3) because FRCC stops any ongoing capacity probes when its time for an RTprop probe.

Assumption 2. Slots align. Slots of flows are the same duration. This is true for ideal link with equal RTprops (§ C.3). On jittery link, slot durations may differ due to differences in the RTTs of the flows. Further, with different propagation delays also slot durations differ. Despite this our analyses yields interesting and empirically valid insights (§ 6.3, § 6.5, § 7).

Assumption 3. Rounds align. Between two `cwnd` updates of a flow, all other flows have a `cwnd` update. For the purpose of analysis, we define rounds at the boundaries of such intervals where each flow probes and updates `cwnd` exactly once. In reality, a flow may probe twice between the probes of another flow, e.g., a flow may probe at the end of one round and at the start of the next round, without other flows updating their `cwnd`s in between. Note, when slot durations are the same, there can be at most two `cwnd` updates of a flow between other flow's `cwnd` updates, as a full round needs to elapse between the first and third `cwnd` update of a flow, during which all other flows will probe at least once.

Assumption 4. Synchronous updates. (We relax this for § 6.4) All `cwnd` updates occur synchronously. I.e., in a round, all flows first update their estimates and then use these to update their `cwnd`. In reality, the `cwnd` updates may happen sequentially, i.e., one flow probes and updates their `cwnd`, resulting a new reference state, then another flow estimates the new state, and then updates its `cwnd`.

Assumption 5. At least one non-colliding cruise before probe. Before every capacity probe, there is at least one cruise measurement that does not collide with a probe of some other flow. This ensures we get an estimate of $\widehat{\texttt{rRTT}_\texttt{i}}$ and $\widehat{\texttt{rtput}_\texttt{i}}$ that is not biased by the probe of other flows (i.e., no impact of (E3)). Note, our design ensures that there is at least one cruise slot before a probe. Further, over a round there will be many cruise slots that do not collide with probe (from the pigeon-hole principle, there are kN_G in steady-state and only N_G flows or probe slots). But all the measurements in the cruise slot before a flow's probe may collide with other probes, we assume that this does not happen.

Assumption 6. Use alternate `cwnd` update. Only for Theorem 6.2, we use the alternate update (line 6) instead of the main update (line 7).

Assumption 7. Known RTprop and one-sided error in RTT, throughput. Only for Theorem 6.2, we assume that there is no error due to (E1/E2) in RTprop, i.e., $\forall i, \widehat{R}_i = R_i$. We also assume that the error due to (E1/E2) in RTT and throughput measurements is one-sided, i.e., RTTs only increase due to noise and throughputs only decrease. This does not hold in reality. Fig. 11 shows that RTTs oscillate and thus

can be both more or less than the fluid reference. Likewise, if one flow's throughput drops because of RTT inflation then the other flows' throughput increases with a closed-loop system.

We believe these assumptions are not needed for Theorem 6.2. We made them historically and hope to remove them. We designed our estimates so that errors vanish with increasing delay and delay variations, and the polarity of the errors has little consequence. For instance, for the capacity estimate, the excess delay is a difference of two potentially erroneous RTT measurements, and we model both positive and negative errors in excess delay. Removing this assumption may change the guidance on design parameter choices. However, empirically, the current design parameters work fine (§ 7).

C.3 Ideal link

C.3.1 Bounds on estimates

LEMMA C.1. *For an ideal link, with equal propagation delays, when the link is utilized, and there are no* `cwnd` *changes, there are no RTT variations due to (E2), i.e., effects of packet interleaving or differences in feedback delay.*

Proof. When the link is utilized, every $1/C$ seconds (i.e., a transmission delay), a packet is dequeued by the bottleneck link, due to the ACK-clock, this dequeue triggers an enqueue exactly R seconds later (as all $R_i = R$). As a result every $1/C$ seconds a packet is enqueued and dequeued, and there is no variation in queuing delay or RTT. Note, this is not true when RTprops are different, which is why we see the oscillations in Fig. 10. □

It suffices for us to look at the execution when the link is already utilized, as FRCC ensures that the bottleneck link is utilized exponentially fast. This is because there is at least one flow in the network, and FRCC increases `cwnd` until, the minimum RTT is $1*\theta$ more than R. As a result, there is always a queuing delay, and link is utilized. So we can use Lemma C.1 in deriving the state transition function. This derivation then works for states that start with a utilized link.

Due to Lemma C.1, since all RTTs are the same, they are equal to $\texttt{rRTT}_\texttt{i}$, and so our RTT estimates are correct: $\widehat{\texttt{rRTT}_\texttt{i}} = \texttt{rRTT}_\texttt{i}$, $\widehat{\Delta d_i} = \Delta d_i$, and $\widehat{R_i} = R_i = R$ (equal RTprops, and draining occurs synchronously (**Assumption 1.**)).

Further, since we measure throughput over packet-timed RTT the $\widehat{\texttt{rtput}}$ is `cwnd`/RTT which is same as the `rtput` `=cwnd/rRTT` on the fluid reference (as all RTTs are equal to `rRTT` due to Lemma C.1).

Note, we used the fact that updates are synchronous in computing these bounds, i.e., the estimates are not impacted by other flows' `cwnd` updates within the round **Assumption 4.** (and consequently **Assumption 3.** and **Assumption 2.**). We also assumed that probes do not collide and that there is a cruise slot before the probe that did not collide with any probe (**Assumption 5.**).

As a result, $\widehat{N_{T_i}} = N_T$, $\widehat{C}_i = C$, $\widehat{N_{C_i}} = N_{C_i}$, i.e., there is no error in estimation compared to reference execution.

C.3.2 Next state, and state transition function

Now, we derive how the state variables evolve over rounds. We re-write the `cwnd` update equations (§ 5.2) given that the estimates are the same as the reference state, and then consider the reference state after all flows have updated their `cwnd`. Note, in reality, flows may update their `cwnd` one by one (but we assume updates are synchronous **Assumption 4.**).

Reference state after a round is (Eq. C.3):

$$N_T[\mathtt{r_F}] = \frac{\sum_i \texttt{cwnd}_\texttt{i}[\mathtt{r_F}] - CR}{C\theta} \tag{C.4}$$

$$N_{C_i}[\mathtt{r_F}] = \frac{\sum_i \texttt{cwnd}_\texttt{i}[\mathtt{r_F}] - CR}{C\theta} \tag{C.5}$$

Substituting next `cwnd`s, (i.e., $\texttt{cwnd}_\texttt{i}[\mathtt{r_F}] = (1-\alpha)\cdot\texttt{cwnd}_\texttt{i}[\mathtt{r}] + \alpha\cdot\texttt{target\_cwnd}_\texttt{i}$, where $\texttt{target\_cwnd}_\texttt{i} = \texttt{cwnd}_\texttt{i}\cdot N_{C_i}/N_T$), i.e., the alternate `cwnd` update (line 6) and simplifying:

$$N_{C_i}[\mathtt{r_F}] = \frac{N_{C_i}((1-\alpha)N_T + \alpha N_G)}{(1-\alpha)N_T + \alpha N_{C_i}} \tag{C.6}$$

$$N_T[\mathtt{r_F}] = (1-\alpha)N_T + \alpha N_G + \frac{\alpha R}{\theta}\left(\frac{N_G}{N_T} - 1\right) \tag{C.7}$$

Note, for the proof on ideal link, we do not need the clamps (δ_H, δ_L). So we just use: $\texttt{target\_cwnd}_\texttt{i} = \texttt{cwnd}_\texttt{i}\cdot N_{C_i}/N_T$. The clamps are needed in the jittery link execution to bound the impact of potentially wrong `cwnd` updates when estimates have errors. Further, in the simplification, we also substituted $\sum_{i=1}^{N_G} \frac{1}{N_{C_i}} = 1$ (follows from the definition of N_{C_i} in Eq. C.2). We get the N_G from $\sum_{i=1}^{N_G} 1$.

C.3.3 State trajectories (alternate `cwnd` update line 7)

We now follow these state update equations to see what happens at round large enough (time large enough). This is visualized in Fig. 14.

LEMMA C.2. *$N_T \to N_G$ exponentially fast (as long as $N_T\theta > R$).*

Proof. Rewriting Eq. C.7, we get:

$$\frac{N_T[\mathtt{r}+1] - N_G}{N_T[\mathtt{r}] - N_G} = (1-\alpha) + \alpha\frac{-R}{\theta N_T[\mathtt{r}]}$$

Based on this equation, if the magnitude of RHS is less than 1 (say $0 < |\text{RHS}| \le \beta < 1$), then the gap between N_T and N_G decays exponentially fast. Here, $|x|$ is the absolute value (magnitude) of x.

$$\left|\frac{N_T[\mathtt{r}+1] - N_G}{N_T[\mathtt{r}] - N_G}\right| = \left|(1-\alpha) + \alpha\frac{-R}{\theta N_T[\mathtt{r}]}\right| \le \beta$$

Substituting $|N_T[\mathtt{r}] - N_G|$ by $x[\mathtt{r}]$,

$$x[\mathtt{r}+1] \le x[\mathtt{r}]\beta$$
$$\implies x[\mathtt{r}+k] \le x[\mathtt{r}]\beta^k$$

$\implies x \to 0$, or $N_T \to N_G$ exponentially fast, as $k \to \infty$

The RHS is average of 1 and a negative number, so it is always less than 1. We just need to show that it is greater than -1. If $N_T\theta > R$, then RHS is > -1. If we set $\theta = 2R$ (we know R from **Assumption 1.**), then we just need $N_T > 1/2$.

Recall, N_T comes from queueing delay in the reference execution. Also recall that $\widehat{N_{C_i}} = N_{C_i} \geq 1$ ($N_{C_i} \geq 1$ from definition of N_{C_i} in Eq. C.2). Whenever $N_T < 1$, the `cwnd` for *all* flows increases as $\texttt{target\_cwnd} = \texttt{cwnd}\frac{N_{C_i}}{N_T} = \texttt{cwnd}\frac{(\geq 1)}{<1} > \texttt{cwnd}$. As a result, the queueing delay and in-turun N_T increase. Once $N_T > 1/2$, it will not decrease below half, as the distance between N_T and N_G always decreases (from the proof above) and N_G is always ≥ 1. □

LEMMA C.3. *Given $N_T = N_G$, $N_{C_i} \to N_G$ exponentially fast.*

Proof. Substituting $N_T = N_G$, and $x[\mathtt{r}] = \frac{N_G}{N_{C_i}[\mathtt{r}]}$ in Eq. C.6, we get:

$$\frac{N_G}{x[\mathtt{r}+1]} = \frac{N_G}{x[\mathtt{r}]}\frac{N_G}{(1-\alpha)N_G + \alpha\frac{N_G}{x[\mathtt{r}]}}$$
$$\implies x[\mathtt{r}+1] = (1-\alpha)x[\mathtt{r}] + \alpha$$
$$\implies x[\mathtt{r}+k] = (1-\alpha)^k x[\mathtt{r}] + (1-(1-\alpha)^k)$$
$$\implies x \to 1, \text{ or } N_{C_i} \to N_G \text{ exponentially fast, as } k \to \infty$$

□

C.3.4 State trajectories (main `cwnd` update line 6)

While $\texttt{target\_cwnd}_\mathtt{i} = \texttt{cwnd}_\mathtt{i} \cdot \widehat{N_{C_i}}/\widehat{N_{T_i}}$ (line 7) was the alternate target `cwnd`, our main target `cwnd` is: $\texttt{target\_cwnd}_\mathtt{i} = \texttt{cwnd}_\mathtt{i} \cdot (R_i + \widehat{N_{C_i}}\theta)/(R_i + \widehat{N_{T_i}}\theta)$.

This update is more stable/robust than the original one. Specifically, this does not require $N_T\theta > R$.

Under the main `target_cwnd` choice , the state variables evolve as (by substituting the new `cwnd` in Eq. C.4 and Eq. C.5):

$$N_T[\mathtt{r}+1] = (1-\alpha)N_T + \alpha N_G \tag{C.8}$$
$$N_{C_i}[\mathtt{r}+1] = N_{C_i}\frac{R + \theta((1-\alpha)N_T + \alpha N_G)}{R + \theta((1-\alpha)N_T + \alpha N_{C_i})} \tag{C.9}$$

LEMMA C.4. *$N_T \to N_G$ exponentially fast.*

Proof. From N_T state update equation (Eq. C.8):

$$N_T[\mathtt{r}+k] = (1-\alpha)^k N_T[\mathtt{r}] + (1-(1-\alpha)^k)N_G$$
$$\implies N_T \to N_G \text{ exponentially fast, as } k \to \infty$$

□

LEMMA C.5. *Given $N_T = N_G$, $N_{C_i} \to N_G$ exponentially fast.*

Proof. In Eq. C.9, substituting $N_T = N_G$, and re-writing:

$$\frac{N_{C_i}[\mathtt{r}+1] - N_G}{N_{C_i} - N_G} = \frac{R + \theta(1-\alpha)N_G}{R + \theta((1-\alpha)N_G + \alpha N_{C_i})}$$

Since $N_{C_i} \geq 1$ (from the definition of N_{C_i}, i.e., Eq. C.2), the RHS is < 1, also RHS > 0 (as all quantities are non-negative). Say $0 < \text{RHS} \leq \beta < 1$, then:

$$\implies \frac{N_{C_i}[\mathtt{r}+1] - N_G}{N_{C_i} - N_G} \leq \beta < 1$$
$$\implies N_{C_i}[\mathtt{r}+k] - N_G \leq (N_{C_i}[\mathtt{r}] - N_G)\beta^k$$
$$\implies N_{C_i} \to N_G \text{ exponentially fast, as } k \to \infty \text{ (as } \beta^k \to 0)$$

□

Note, since $\alpha = 1$, without clamps, FRCC actually converges N_T in 1 round with the main update and N_{C_i} in 1 round with the alternate update.

C.4 Jittery link

C.4.1 Bounds on estimates

We compute bounds on the estimates made by FRCC, assuming each RTT can increase by up to $\Delta R/2$ and throughput can decrease by γ_R due to (E1/E2) (**Assumption 7.**). As a result, the error in a RTT difference measurement is $\pm\Delta R$ (since $(R_1 \pm \Delta R/2) - (R_2 \pm \Delta R/2) = (R_1 - R_2) \pm \Delta R$).

In these estimates we do not incorporate impact of (E3). We assume there is a cruise measurement before probe that does not overlap with other probes (**Assumption 5.**), hence (E3) errors for cruise slot estimates ($\widehat{\texttt{rtput}}$, $\widehat{\texttt{rRTT}}$, $\widehat{R}$) get filtered out by max/min. The theorem statement also assumes probes do not collide so (E3) does not affect capacity estimate.

1. $\widehat{R} = R$. We have assumed that $\widehat{R}$ is correct (no error) (**Assumption 1.**, **Assumption 7.**).
2. $\widehat{\texttt{rRTT}_\mathtt{i}} \in [\texttt{rRTT}_\mathtt{i}, \texttt{rRTT}_\mathtt{i} + \Delta R/2]$.
3. $\widehat{\texttt{rtput}_\mathtt{i}} \in [\texttt{rtput}_\mathtt{i}/\gamma_R, \texttt{rtput}_\mathtt{i}]$. This implies that $\frac{C}{\widehat{\texttt{rtput}_\mathtt{i}}} \in [N_{C_i}, N_{C_i}\gamma_R]$ as $C/\texttt{rtput}_\mathtt{i} = N_{C_i}$. We will use this later.
4. $\widehat{\Delta d_i} \in [E_i/(C+\Delta R), E_i/(C-\Delta R)]$. (For $\widehat{C}_i$ in $\widehat{N_{C_i}}$).
5. $\widehat{N_{T_i}} = (\widehat{\texttt{rRTT}_\mathtt{i}} - R_i)/\theta \in [N_T, N_T + \Delta R/2\theta]$
6. Error in current flow count estimate:

$$\widehat{N_{C_i}} = \frac{\widehat{C}_i}{\widehat{\texttt{rtput}_\mathtt{i}}} = \frac{E_i}{\widehat{\Delta d_i}}\frac{1}{\widehat{\texttt{rtput}_\mathtt{i}}} = \frac{\gamma\widehat{\texttt{rtput}_\mathtt{i}}\widehat{N_{T_i}}\mathcal{D}}{E_i/C \pm \Delta R}\frac{1}{\widehat{\texttt{rtput}_\mathtt{i}}}$$
$$= \frac{\gamma\widehat{N_{T_i}}\mathcal{D}}{\gamma\widehat{\texttt{rtput}_\mathtt{i}}\widehat{N_{T_i}}\mathcal{D}/C \pm \Delta R}$$

When $\mathcal{D} \geq \Delta R$,

$$= \frac{\gamma\widehat{N_{T_i}}}{\gamma\widehat{\texttt{rtput}_\mathtt{i}}\widehat{N_{T_i}}/C \pm 1}$$

Substituting $C/\widehat{\texttt{rtput}_\mathtt{i}}$ derived in step 3:

$$\widehat{N_{C_i}} \in \left[N_{C_i}\frac{\gamma\widehat{N_{T_i}}}{\gamma\gamma_R\widehat{N_{T_i}} + N_{C_i}}, N_{C_i}\frac{\gamma\widehat{N_{T_i}}}{\gamma\widehat{N_{T_i}} - N_{C_i}}\right] \tag{C.10}$$

Note, we used the fact that updates are synchronous in computing these bounds, i.e., the estimates are not impacted by other flows' `cwnd` updates within the round **Assumption 4.** (and consequently **Assumption 3.** and **Assumption 2.**).

Note, the upper bound in Eq. C.10 may be very big (e.g., when N_{C_i} is close to $\gamma\widehat{N_{T_i}}$). The clamps on $\widehat{N_{C_i}}$ allow us to manage the impact of errors during such situations.

C.4.2 Next state, state transition function

We re-write the `cwnd` update equations (using the alternate update line 1, using **Assumption 6.**) given the errors in estimates, and then consider the stabilized state after all flows have updated their `cwnd` (**Assumption 4.**). In reality, not all flows may update their `cwnd` in a round, and the flows may not make measurements and updates in lock-step.

Consider the next reference state (Eq. C.3). We plug the values of $\texttt{cwnd}_i[\mathtt{r_F}]$ from line 1 in Eq. C.4 and Eq. C.5, and simplifying, we get the following state update equations. Note, these equations are in terms of $\widehat{N_{C_i}}$, this term is after the clamps have been applied.

$$N_{C_i}[\mathtt{r_F}] = \frac{\sum_j(\prod_{k\neq j}N_{C_k}\widehat{N_{T_k}})((1-\alpha)\widehat{N_{T_j}}+\alpha\widehat{N_{C_j}})}{(\prod_{k\neq i}N_{C_k}\widehat{N_{T_k}})((1-\alpha)\widehat{N_{T_i}}+\alpha\widehat{N_{C_i}})} \tag{C.11}$$

$$\begin{aligned} N_T[\mathtt{r_F}] &= \frac{R+\theta N_T}{\theta}\left((1-\alpha)\sum_j\frac{1}{N_{C_j}}+\alpha\sum_j\frac{\widehat{N_{C_j}}}{N_{C_j}}\frac{1}{\widehat{N_{T_j}}}\right)-\frac{R}{\theta} \\ &= \frac{R+\theta N_T}{\theta}\left((1-\alpha)+\alpha\sum_j\frac{\widehat{N_{C_j}}}{N_{C_j}}\frac{1}{\widehat{N_{T_j}}}\right)-\frac{R}{\theta} \end{aligned} \tag{C.12}$$

We verified our algebraic steps using sympy [32]. To get Eq. C.12, we used $\sum_{i=1}^{N_G}\frac{1}{N_{C_i}}=1$ (from the definition of N_{C_i}, i.e., Eq. C.2). Also observe that if we substitute $\widehat{N_{T_i}}=N_T$, and $\widehat{N_{C_i}}=N_{C_i}$, we get the ideal link equations (Eq. C.6 and Eq. C.7).

For two flows, these equations simplify to:

$$N_{C_1}[\mathtt{r_F}] = 1+\frac{N_{C_1}\widehat{N_{T_1}}((1-\alpha)\widehat{N_{T_2}}+\alpha\widehat{N_{C_2}})}{N_{C_2}\widehat{N_{T_2}}((1-\alpha)\widehat{N_{T_1}}+\alpha\widehat{N_{C_1}})} \tag{C.13}$$

$$N_T[\mathtt{r_F}] = \frac{R+\theta N_T}{\theta}\left((1-\alpha)+\alpha\left(\frac{\widehat{N_{C_1}}}{N_{C_1}\widehat{N_{T_1}}}+\frac{\widehat{N_{C_2}}}{N_{C_2}\widehat{N_{T_2}}}\right)\right)-\frac{R}{\theta} \tag{C.14}$$

These equations describe the reference state in the next round as a function of the reference state in the current round and estimates. In the previous section (§ C.4.1), we had derived bounds on estimates relative to the reference state. Using these, we obtain an expressions showing bounds on the reference state in the next round $\langle N_{C_i}[\mathtt{r_F}], N_T[\mathtt{r_F}]\rangle$ in terms of the reference state in the current round $\langle N_{C_i}, N_T\rangle$, and design/network parameters, eliminating all estimates (e.g., $\widehat{N_{C_i}}$,$\widehat{C_i}$,$\widehat{\texttt{rtput}_\texttt{i}}$,$\widehat{\texttt{rRTT}_\texttt{i}}$, etc.). We do not show these expressions for brevity.

C.4.3 State trajectories. Lemmas in Theorem 6.2

We encode the transition function into Z3 to prove lemmas about state evolution over rounds. We ask Z3 to find a counterexample to our lemma. A counterexample is an assignment to the variables, i.e., current state ($\langle N_{C_i}, N_T\rangle$), design parameters ($\mathcal{D}$, θ) and network parameters (C, R, D, γ_R), for which our lemma evaluates to `False`. If Z3 outputs `UNSAT`, i.e., there is no counterexample, then our lemma is `True`. We also used Z3 to come up with the lemmas to begin with. This is an iterative trial and error process where we ideate lemmas, get potential counterexamples and update the lemmas. We describe this process more in § C.4.4.

We list the lemmas part of Theorem 6.2. We prove these lemmas for the design parameters: $\gamma=4$, $\alpha=1$, $\delta_H=1.3$, $\delta_L=1.25$. Our design works whenever the network parameters satisfy: $\mathcal{D}\geq\Delta R$, $\theta*N_G>R$, $\theta>\Delta R/2$, $\gamma_R\leq 2$. Note, $\theta*N_G>R$ was also needed when analyzing the less stable `cwnd` update in § 6.2. This was not needed for the more stable `cwnd` update. We believe this will also not be needed for the jittery link for the more stable `cwnd` update.

For readability, we use the following substitutions in our lemmas:

$$\begin{aligned} &N_{C_i}^{\mathtt{I}}=N_{C_i}[\mathtt{r}] && N_{C_i}^{\mathtt{F}}=N_{C_i}[\mathtt{r}+1] \\ &N_{C_i}^{\mathtt{LB}}=\frac{N_T^{\mathtt{UB}}}{N_T^{\mathtt{UB}}-1}=6.6/5.6 && N_{C_i}^{\mathtt{UB}}=6.6 \\ &N_T^{\mathtt{I}}=N_T[\mathtt{r}] && N_T^{\mathtt{F}}=N_T[\mathtt{r}+1] \\ &N_T^{\mathtt{LB}}=0.15 && N_T^{\mathtt{UB}}=3.3 \\ &N_T^{\mathtt{LB-BAND}}=1.8 && N_T^{\mathtt{UB-BAND}}=2.2 \\ &\sigma=1.01 \end{aligned}$$

We obtained these non-trivial constants using binary search. We find the most extreme values at which the lemmas continue to hold. E.g., if we decrease $N_{C_i}^{\mathtt{UB}}$ from 6.6 to 6.5, Lemma C.6 breaks. Note, these are worst-case performance bounds, i.e., over worst-case jitter patterns. Empirical performance of FRCC is better § 7. Further, the lemmas also only describe worst-case progress, e.g., N_T may only show minor progress ($\sigma=1.01$) when it is already close to converging, but it shows major progress when it is far from converging because of asymmetry in our capacity estimate (§ 5.3.1).

Note, these lemmas are for a two flow system ($N_G=2$) with flows f_i and f_j. Since $N_{C_j}=\frac{N_{C_i}}{N_{C_i}-1}$ (from the definition of N_{C_i} in Eq. C.2, along with $N_G=2$), N_{C_i} represents the state of both the flows, and we state the lemmas only in terms of N_{C_i} without losing generality. The bounds $N_{C_i}^{\mathtt{LB}}$ and $N_{C_i}^{\mathtt{UB}}$ also satisfy $N_{C_i}^{\mathtt{UB}}=\frac{N_{C_i}^{\mathtt{LB}}}{N_{C_i}^{\mathtt{LB}}-1}$. As a result, $N_{C_i}\in[N_{C_i}^{\mathtt{LB}},N_{C_i}^{\mathtt{UB}}]$ if and only if $N_{C_j}\in[N_{C_i}^{\mathtt{LB}},N_{C_i}^{\mathtt{UB}}]$.

The lemmas also assume that $N_T>0$ always. When $N_T<N_T^{\mathtt{LB}}=0.15$, Lemma C.7 shows that N_T increases multiplicatively, so as long as we start with $N_T>0$, $N_T>0$ will keep remaining `True`. $N_T>0$ is true whenever the bottleneck queue occupancy is non-zero (this also ensures that the bottleneck link is utilized). It is okay to assume $N_T>0$ because FRCC increases `cwnd` whenever $\widehat{N_{T_i}}<1$

(because there is always at least one flow in the system). Since $\widehat{N_{T_i}} \leq N_T + \Delta R/2\theta$ (§ C.4.1), as long as $\Delta R/2\theta < 1$ (which is what we are designing for), $\widehat{N_{T_i}}$ will always be < 1, whenever $N_T \leq 0$, so FRCC will keep increasing `cwnd` (multiplicatively), so that $N_T > 0$ happens exponentially fast.

LEMMA C.6 (N_T and N_{C_i} bounded). *If initial N_T and N_{C_i} are converged to a bounded range, then they remain converged. I.e.,*

$$N^I_{C_i} \in [N^{LB}_{C_i}, N^{UB}_{C_i}] \implies N^F_{C_i} \in [N^{LB}_{C_i}, N^{UB}_{C_i}] \text{ and,}$$
$$N^I_T \in [N^{LB}_T, N^{UB}_T] \implies N^F_T \in [N^{LB}_T, N^{UB}_T]$$

LEMMA C.7 (N_T converges). *If initial N_T is outside its bounded range, it converges towards the range exponentially fast without overshooting (e.g., if it is above the range it does not reduce to a value below the range), I.e.,*

$$N^I_T > N^{UB}_T \implies \left(N^F_T \leq N^I_T/\sigma \vee N^F_T \leq N^{UB}_T\right) \wedge N^F_T \geq N^{LB}_T \text{ and,}$$
$$N^I_T < N^{LB}_T \implies \left(N^F_T \geq N^I_T\sigma \vee N^F_T \geq N^{LB}_T\right) \wedge N^F_T \leq N^{UB}_T$$

LEMMA C.8 (N_{C_i} converges). *If either the initial N_T or initial N_{C_i} is not converged, then either both converge or two things happen: (1) at least one of them (one that is not already converged) improves (exponentially fast without overshooting) while (2) neither moves away from the bounded range. I.e.,*

$$\begin{aligned}
&\neg(N^I_{C_i} \in [N^{LB}_{C_i}, N^{UB}_{C_i}] \wedge N^I_T \in [N^{LB}_T, N^{UB}_T]) \\
\implies &(N^F_{C_i} \in [N^{LB}_{C_i}, N^{UB}_{C_i}] \wedge N^F_T \in [N^{LB}_T, N^{UB}_T]) \\
&\vee ((\textit{bad } N_{C_i} \textit{ improves} \vee \textit{bad } N_T \textit{ improves}) \\
&\quad \wedge N_{C_i} \textit{ does not degrade} \\
&\quad \wedge N_T \textit{ does not degrade})
\end{aligned}$$

*Where "bad N_{C_i} improves" means that N_{C_i} is outside its converged range, ~~implies~~ **and** it moves exponentially fast towards the range without overshooting, i.e.,*

$$N^I_{C_i} > N^{UB}_{C_i} \wedge N^F_{C_i} \geq N^{LB}_{C_i} \wedge (N^F_{C_i} \leq N^I_{C_i}/\sigma \vee N^F_{C_i} \leq N^{UB}_{C_i}) \text{ or,}$$
$$N^I_{C_j} > N^{UB}_{C_i} \wedge N^F_{C_j} \geq N^{LB}_{C_i} \wedge (N^F_{C_j} \leq N^I_{C_j}/\sigma \vee N^F_{C_j} \leq N^{UB}_{C_i})$$

*"bad N_T improves" means that N_T is outside the $[N^{LB-BAND}_T, N^{UB-BAND}_T]$ band, ~~implies~~ **and** it moves towards it exponentially fast without overshooting:*

$$\begin{aligned}
&N^I_T > N^{UB-BAND}_T \wedge N^F_T \geq N^{LB-BAND}_T \wedge \\
&\qquad (N^F_T \leq N^I_T/\sigma \vee N^F_T \leq N^{UB-BAND}_T) \text{ or,} \\
&N^I_T < N^{LB-BAND}_T \wedge N^F_T \leq N^{UB-BAND}_T \wedge \\
&\qquad (N^F_T \geq N^I_T * \sigma \vee N^F_T \geq N^{LB-BAND}_T)
\end{aligned}$$

"N_{C_i} does not degrade" means that if N_{C_i} is not converged then it does not move away from its converged range, i.e.,

$$(N^I_{C_i} < N^{LB}_{C_i} \implies N^F_{C_i} \geq N^I_{C_i}) \wedge (N^I_{C_i} > N^{UB}_{C_i} \implies N^F_{C_i} \leq N^I_{C_i})$$

Similarly, "N_T does not degrade" means:

$$(N^I_T < N^{LB}_T \implies N^F_T \geq N^I_T) \wedge (N^I_T > N^{UB}_T \implies N^F_T \leq N^I_T)$$

In "bad N_{C_i} improves", you will notice that both the conjunctions use the upper bound. Instead of writing the second conjunction in terms of the lower bound, we write it in terms of N_{C_j} which refers to the other flow. If N_{C_i} is below its lower bound then N_{C_j} has to be above the upper bound. This follows from $N_{C_j} = \frac{N_{C_i}}{N_{C_i}-1}$ (from the definition of N_{C_i} in Eq. C.2, along with $N_G = 2$), and $N^{UB}_{C_i} = \frac{N^{LB}_{C_i}}{N^{LB}_{C_i}-1}$ (from the constant values). We wrote it this way because the relative change in N_{C_i} is higher when $N_{C_i} \geq 1 \geq N_{C_j}$, (i.e., the flow that is consuming lesser fraction of the link has higher relative change in their fraction after the round updates).

We need the "neither degrades" part to ensure that the N_{C_i} and N_T do not oscillate in a way where one of them improves and the other degrades and vice versa without making any progress towards convergence. Similarly, we also need the "one which is not converged improves", or "bad" in "bad N_T improves". If we replace "bad N_T improves", with just "N_T improves", we would get "N_T above the $[N^{LB-BAND}_T, N^{UB-BAND}_T]$ band implies it decreases and N_T below the band implies it increases". This formula is true when N_T is inside the band, and as a result satisfies the post condition of the lemma without forcing N_{C_i} to ever converge towards its range.

Summary. Going back to Fig. 15, if N^I_T is outside its converged range (top or bottom) then it eventually converges from Lemma C.7. I.e., the system is in the left, center, or right boxes in Fig. 15. If state is in the center box, we are done (i.e., system is in steady-state, it stays that way from Lemma C.6 and we deliver the steady-state performance bounds). Left and right boxes are symmetric (one of N_{C_i} or N_{C_j} lies in the right box if the system is in left or right boxes). Without loss of generality, say N_{C_i} is in the right box. Then from Lemma C.8, if N_{C_i} does not improve, then N_T must move towards the narrow band if it is outside the band, whenever it is inside the band, N_{C_i} must improve. N_T can exit the narrow band but not leave its converged range, and the process repeats. N_{C_i} can never move away from the converged range. The result of these movements is that eventually N_{C_i} must converge. These movements are shown using blue arrows in Fig. 15.

All the movements are multiplicative without any overshoots, thus asymptotically the system converges to steady-state in logarithmic steps in the total amount of movement. Each step is a round that lasts for $O(N_G)$ slots, with each slot being 4RTTs, and the total movement is bounded by the steady-state $\texttt{cwnd} = C/N_G * (R + \theta N_G) = \texttt{BDP}/N_G + C\theta N_G$ (this is `cwnd` FRCC maintains in steady-state on ideal link). Giving a total convergence time of $O(N_G \log(\texttt{BDP}))$ RTTs.

Note, N_T may enter and exit the band multiple times, these are all hidden in the big-O notation.

C.4.4 How we come up with the lemmas

Coming up with the lemmas is an iterative process. We start by identifying a region in the state space such that if initial state is in the region, then the final state is in the region. This gives us the shaded region. A rectangular region worked for us. Note, the proof is not tight, the real region could be a different shape. Then we ideate lemmas to describe motions the state makes then it is outside the shaded region. Z3 may prove that the lemma is correct, or give us a counterexample showing how the state moves instead of what our lemma described. Based on this feedback, we update the lemma and iterate.

D Fluid model analysis (different RTprops and multiple bottlenecks)

Different RTprops. We compute value of biased excess delay from the fluid model as follows. Say in the current state, the RTTs (not RTprops) of the two flows are R and ρR (i.e., RTT ratio of ρ), and throughputs are fC and $(1-f)C$. Consequently, the cwnds of the flows are: $\texttt{cwnd}_1 = R \cdot fC, \texttt{cwnd}_2 = R \cdot (1-f)C$ (using $\texttt{throughput} = \texttt{cwnd}/\texttt{RTT}$). When flow f_1 probes, the delay increase seen by the probe Δd is given by:

$$\frac{\texttt{cwnd}_1 + E}{R + \Delta d} + \frac{\texttt{cwnd}_2}{\rho R + \Delta d} = C \tag{D.1}$$

I.e., the RTTs of both flows increase by Δd and the sum of the throughputs during the probe are equal to C. We substitute the cwnds and solve for Δd as a function of ρ, R, E ($\gamma\mathcal{D}$), f, and C. This expression is complicated and we do not show it. But $\Delta d \neq E/C$. Note, if we substitute $\rho = 1$ in Eq. D.1, then $\Delta d = E/C$. If $\rho > 1$ (short RTprop flow probes), $\Delta d > E/C$, while $\Delta d < E/C$, when $\rho < 1$ (long flow probes).

Since Δd is a complicated expression we not substitute it all the way through to get an updated transition function, instead we solve for the fixed-point indirectly. Specifically, the system is in steady-state when no flow has incentive to change its cwnd, i.e., $\widehat{N_{T_i}} = \widehat{N_{C_i}}$. Since flows measure the same queueing delay, $\widehat{N_{T_1}} = \widehat{N_{T_2}}$. Thus, steady-state happens when $\widehat{N_{C_1}} = \widehat{N_{C_2}}$. We substitute the biased Δd in $\widehat{N_C}$ ($= \widehat{C}/\widehat{\texttt{rtput}} = E/(\widehat{\Delta d}\widehat{\texttt{rtput}})$). We get $\widehat{N_{C_i}}$ as a function of the system state and design/network parameters. Solving $\widehat{N_{C_1}} = \widehat{N_{C_2}}$, gives us the fixed-point state as a function of design/network parameters. Note, in the fluid model, hats ($\widehat{N_{C_i}}$) and non-hat (N_{C_i}) variables are the same.

Solving $\widehat{N_{C_1}} = \widehat{N_{C_2}}$ requires finding roots of a high degree polynomial, we use numerical methods (secant method in sympy [32]) to approximate the root.

Multi-bottleneck parking lot topology. The setup and computation of the fixed-point is similar to the different RTprop case. We write fluid model equations for each hop in the parking lot. This time, the excess delay, Δd, when the long flow (f_0 in Fig. 17) probes is given by:

$$\frac{\texttt{cwnd}_0 + E}{R_0 + \Delta d} + \frac{\texttt{cwnd}_1}{R_1 + \Delta d/\texttt{hops}} = C \tag{D.2}$$

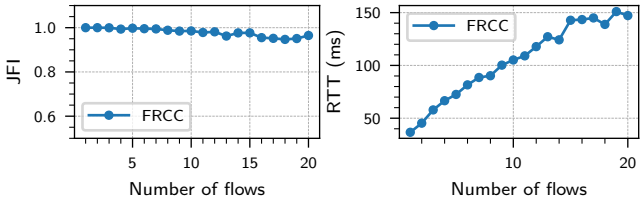

Figure 25: FRCC with more flows.

The main difference is that if before the probe the RTTs of flow f_0 (long flow) and f_1 (short flow) were R_0 and R_1, then after the probe the RTTs become $R_0 + \Delta d$ and $R_1 + \Delta d/\texttt{hops}$ respectively. I.e., the increase in the RTT of the short flow is $1/\texttt{hops}$ times less than the increase in the RTT of the long flow. This is because the RTTs of all the short flow (f_1 to f_{hops}) increases and f_0 witnesses the sum of the delays over all hops. The delay at each hops is the same (due to symmetry fluid equations for each hop). So the total increase in RTTs of all the short flows matches the increase in RTT of the long flow.

When solving for the fixed-point, $\widehat{N_{T_i}} = \widehat{N_{C_i}}$. Due to the same reason as above, this time, $\widehat{N_{T_0}} = \texttt{hops}\widehat{N_{T_1}}$. So we solve for $\widehat{N_{C_0}} = \texttt{hops}\widehat{N_{C_1}}$ to determine the fixed-point.

E Supplementary empirical evaluation

Fig. 26 and Fig. 27 complement Fig. 1 and Fig. 2 to show that BBRv3 and Reno also cause starvation in the scenarios with different propagation delays and jitter.

Impact of upper clamp with higher flow counts. Fig. 25 shows FRCC with more flows. The upper clamp on slot count is $K_{\max} = 20$, and the dynamic slot count is $k\widehat{N_T}$, with $k = 2$. With up to 10 flows the upper clamp is not hit and JFI is close to 1. Beyond that there are less than kN_G slots and collisions become more likely creating some unfairness.

Note, since FRCC's practical convergence speed is slow, to ensure flows reach steady-state, we ran this experiment with RTprop of 10 ms instead of our default of 50 ms (§ 7).

F Glossary

Table 1 and Table 2 parameter and state variable definitions.

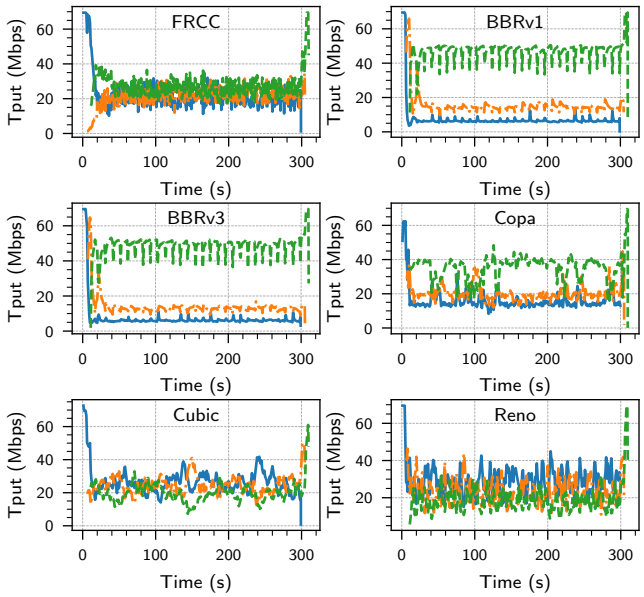

Figure 26: Flows with round-trip propagation delay (Rtprop or R) of 10, 20 and 30 ms. BBR starves the (blue) flow with lower Rtprop.

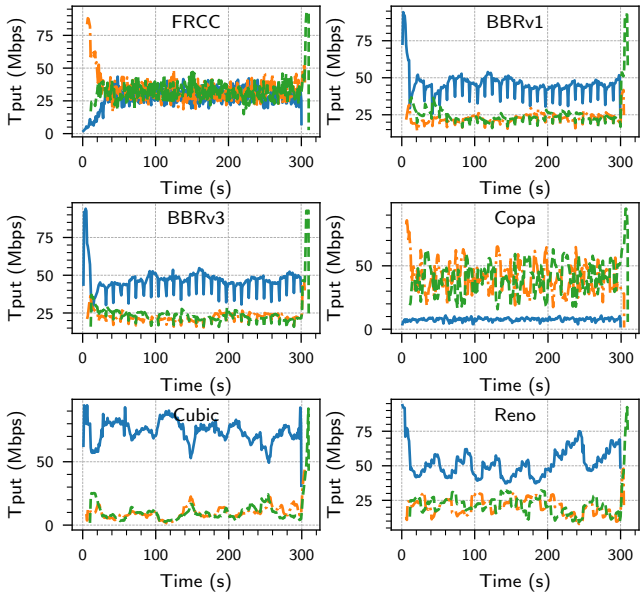

Figure 27: Three flows with $R = 32$ ms. The blue flow experiences 32 ms of ACK aggregation. Cubic and Copa starve flows.

Symbol	Unit	Domain	Description
Network parameters			
C	r	$\mathbb{R}_{>0}$	Link capacity
BDP	p	$\mathbb{R}_{>0}$	Bandwidth delay product
R_i, R	s	$\mathbb{R}_{>0}$	Round-trip propagation delay
D_i, D	s		Maximum network jitter per-packet
N_G	u	$\mathbb{Z}_{\geq 1}$	Number of flows (ground truth)
ΔR	s		Max (additive) error in RTT difference
γ_R	u	$\mathbb{R}_{\geq 1}$	Max (multiplicative) error in Tput
Design parameters			
R_{max}	s	$\mathbb{R}_{>0}$	Assumption about maximum R
$\mathcal{D}$	s	$\mathbb{R}_{>0}$	Amount of ΔR we want to tolerate
θ	s	$\mathbb{R}_{>0}$	Delay we maintain per flow
γ	u	$\mathbb{R}_{>1}$	Gain multiplier for probing
α	u	$(0,1]$	cwnd update weight
δ_L, δ_H	u	$\mathbb{R}_{>1}$	Clamps for cwnd update
k	u	$\mathbb{R}_{>1}$	$k\widehat{N_T}$ = Number of slots in a round
T_P	s	$\mathbb{R}_{>0}$	Probe duration (= 1 RTT)
T_R	s	$\mathbb{R}_{>0}$	Time between RTprop probes (= 30 s)

Table 1: Glossary of parameters. For the units: r = rate (i.e., packets/seconds), s = seconds, p = packets, u = unit-less. By default, the domain is $\mathbb{R}_{\geq 0}$. Subscript i denotes quantity seen/maintained by flow i. We drop the subscripts when clear from context.

Symbol	Unit	Domain	Description
$\texttt{cwnd}_i$	p	$\mathbb{R}_{\geq 1}$	Congestion window size
$\widehat{C}_i$	r		Link capacity estimate
$\widehat{R}_i$	s		R estimate
$\texttt{rRTT}_i, \widehat{\texttt{rRTT}_i}$	s		RTT in a round
$\texttt{rtput}_i, \widehat{\texttt{rtput}_i}$	r		Throughput in a round
$N_T, N_{T_i}, \widehat{N_{T_i}}$	u		Target flow count
$N_{C_i}, \widehat{N_{C_i}}$	u	$N_{C_i} \geq 1$, $\widehat{N_{C_i}} \geq 0$	Current flow count (from the perspective of flow i)
E_i	p	$\mathbb{R}_{\geq 1}$	Excess packets sent by probe
$\Delta d_i, \widehat{\Delta d_i}$	s	$\Delta d_i \geq 0$, $\widehat{\Delta d_i} \in \mathbb{R}$	Excess delay due to probe

Table 2: Glossary of state variables. We use hats ($\widehat{\texttt{rRTT}_i}$) to denote estimates made by FRCC, and no annotation ($\texttt{rRTT}_i$) to denote the value of the state variable in the fluid reference execution. We use a suffix of $[\texttt{r}]$ to denote the round number (e.g., $\texttt{rRTT}_i[\texttt{r}]$, $\widehat{\texttt{rRTT}_i}[\texttt{r}]$). See Table 1 for convention on units, domain, and subscript i.