No Signal to Rule Them All: A Systematic Analysis of In-Network Congestion Signals

Sarah McClure
UC Berkeley, USA

Nandita Dukkipati
Google, USA

Sylvia Ratnasamy
UC Berkeley, USA

Scott Shenker
UC Berkeley, USA
ICSI, USA

Abstract

In this paper, we address the following question: *what in-network signals should a network provide to congestion control algorithms?* To answer this guiding question, we use prior work to automatically generate congestion control algorithms optimized for a given performance objective and set of in-network congestion signals. We then make observations about the relative value of these congestion signals across a range of performance objectives. Our analysis yields a surprising central finding: for the average case, sophisticated In-Network Telemetry (INT) offers minimal performance benefits over traditional end-to-end (E2E) signals, with performance typically within 3%. We also find no single "best" INT signal, but rather a clear trade-off that manifests in many scenarios: link-based signals often excel at controlling delay, while queue-based signals are better for maximizing throughput. To make these findings concrete, we validate them by examining the extent to which in-network signals improve the performance of the BBR congestion control algorithm.

2012 ACM Subject Classification Networks → Network protocol design; Networks → Signaling protocols; Networks → Transport protocols; Networks → Network control algorithms; Networks → Network simulations; Networks → Network performance modeling; Networks → Network performance analysis

Keywords and phrases Congestion control, in-network telemetry

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.12

Related Version *Extended Version*: PDF

Supplementary Material *Software*: https://github.com/smcclure20/ns3-signal-eval
Software: https://github.com/smcclure20/rplus

1 Introduction

Because it plays such an important role in the resulting performance of individual flows, congestion control is a mainstay of the networking literature, with new congestion control algorithms appearing in almost every networking conference. But even if we look only at widely deployed congestion control algorithms, rather than including research proposals, the field has made impressive progress over the years. There have been several generations of widely deployed congestion control algorithms, starting with loss-based congestion control algorithms that focused on the WAN [19, 40, 36] which were followed by congestion control algorithms – most notably BBR [10] – that utilized delay and throughput rather than loss as the primary congestion signal. When datacenters became prevalent, with their short

RTTs and emphasis on tail latencies, a new generation of congestion control algorithms was deployed, starting with DCTCP [5] and leading to more recent algorithms [30, 24, 28, 48].

One constant throughout this ongoing evolution was that widely deployed congestion control algorithms were mostly based on end-to-end (E2E) signals, such as RTT, packet drops, and observed throughput. The one obvious exception to the end-to-end nature of the signals used by these congestion control algorithms was ECN, whose origins date back to DECBit [34]. In the WAN context, ECN provided limited value since it was not universally adopted. Thus, congestion control algorithms had to expect a mixture of ECN-enabled and ECN-incapable routers along the path. But in the datacenter context, because uniform ECN adoption could be achieved, DCTCP leveraged ECN's flexibility to provide a richer form of feedback from individual routers.

We have recently entered a new era in datacenter congestion control, since a much richer set of router-based signals is becoming available from individual routers. These newly emerging signals, which are referred to as *in-network telemetry* (INT), are collected at routers and made available to hosts via fields in packet headers, as documented in the CSIG Internet Draft [35]. There are many possible signals that could be generated at individual routers, but we must be careful about which ones the field eventually adopts, for three reasons. First, not all INT signals can be reported to a congestion control algorithm, because header real estate is scarce. Second, the mechanisms for collecting these signals operate at packet speeds and therefore must be embedded in router hardware/firmware. Thus, vendors must make a choice about which to support rather than enabling each customer to choose whatever signals they want. Lastly, we expect that some INT signals will be better than others, in the sense of leading to better datacenter congestion control algorithms. Thus, we must make a choice about INT signals, and that choice matters.

To provide some guidance on this choice, in this paper we address the question: *what INT signals would be most valuable for datacenter congestion control algorithms (CCAs)?* But to answer this question fully and fundamentally, we must take a general approach that does not start with limiting assumptions about the particular context in which these INT signals will be used. More specifically, we do not want to fix on a given CCA, or on a particular performance goal, or on a given workload. Instead, we want to understand how the choice of INT signals might be impacted by these various factors and whether there are any general lessons to be learned. Such lessons will allow us to ask what INT signals are best for CCAs that are designed from the ground up to leverage those signals, which is the question we care about in the long-term.

Of course, others have looked at how to leverage INT signals. In particular, as we discuss in §8, prior work has explored co-designing INT and CCAs for specific performance metrics or workloads (e.g., HPCC [28]), or optimizing a specific CCA to use a specific INT signal [41], but we are unaware of work that attempts to answer the question as generally as we tried to do.

This generality poses a methodological problem, because the value of a particular INT signal is fundamentally dependent on how well a CCA utilizes it. Ideally, we would evaluate a set of INT signals S based on the performance of a CCA that was optimally designed to leverage that set of INT signals. And, given our long experience with congestion control algorithms, we expect that the "optimal" CCA will depend on both the set of INT signals and the desired performance objectives (and in what follows we assume that these objectives are some function of the resulting throughput and delay). But we have no known methods for generating the optimal CCAs for a specific set of signals and performance objectives.

As a pragmatic way out of this methodological bind, we propose to leverage prior work on *learned* congestion control algorithms. These learning-based techniques search the space of congestion control algorithms that have access to a set of INT signals, S, during training and return the optimally performing algorithm discovered during the search process. Our proposal is to use this best learned algorithm – denoted lCCA(S) – as a stand-in for the optimal datacenter congestion control algorithm that uses S. This turns the unattainable ideal of optimality into the tangible result of learning, which we think will suffice for our efforts.

While one might leverage any learning-based approach, for reasons we discuss in §3, we build on one of the earliest approaches to learning-based congestion control – Remy [42] – and add several key modifications to improve its search capabilities in our context. We make no claim that our learning technique, which we call R+, is the best, but we find that the algorithms it produces – the lCCAs – generally match or outperform both the state-of-the-art hand-crafted CCAs and other learned CCAs that we obtaining from other learning algorithms. Thus, our research approach is to take, for any given set of INT signals S, the resulting lCCA(S) as a "tractable approximation of the optimal" [44] and use it to evaluate the benefit of the set of signals S.

Note that the resulting lCCAs are *not* candidates for deployment. We recognize that learned CCAs often do not generalize beyond their training environment, and the world of datacenters environments is far broader than what our test cases encompass. But we are comparing lCCAs within the same environments that we use for training, so we believe that the basic comparisons will still provide insights, even if the resulting lCCAs may not sufficiently generalize to all settings they might encounter in real deployments.

The bulk of this paper is devoted to a series of experiments comparing the performance of the resulting lCCA(S) as we vary the set of signals and the performance objectives, over a range of datacenter network scenarios. For the range of network configurations and performance objectives that we investigate, our core findings are twofold:

(1) CCAs that exploit INT signals see little average improvement over those that use just the E2E signals of delay, loss, sending rate, and receiving rate. More specifically, the best performance with INT is on average within 3% of that achieved using only E2E signals.

(2) There is no clear "winner" amongst the set of INT signals. On average, the performance they achieve is within 1% of each other.

These core findings were a surprise to us, and hence the majority of this paper is about understanding them more deeply. Exploring the first finding leads to the following underlying results:

1a **Delay and send/receive rates suffice.** We find that finding (1) would not hold if loss was the primary E2E congestion signal (as was the case in the past); for example, the best INT dominates loss alone by 30% on average, but other E2E signals such as RTT and send/receive rates close the gap.

1b **While we typically average over a range of network scenarios, the specifics of a network scenario occasionally matters to a modest degree.** Finding (1) holds in part because we are averaging across network scenarios, but there are specific scenarios where INT offers modest benefits. For instance, for one of our performance objectives (that values delay and throughput roughly equally), INT offers up to 7% higher performance in up to 11% of scenarios. These gains can be relevant for performance-sensitive datacenter contexts with specific operating regimes.

1c **INT typically helps the most in extreme network scenarios.** Investigating *when* INT wins, we find that these specific scenarios are typically characterized by having extreme (high or low) levels of load or numbers of senders. Our hypothesis is that, in such cases, the E2E signals for an individual flow capture only a partial view of router conditions, while INT signals always capture a more "global" view of router performance, and this gives INT an edge. Validating this hypothesis remains an open question, which we hope to return to in future work.

1d **In these extreme scenarios, INT is a better "multitasker".** Our investigation of *why* INT wins in the above scenarios reveals that, when considering performance objectives that optimize for both throughput and delay but prioritize one over the other (e.g., weighting delay as twice as important than throughput), INT is comparable to E2E signals in optimizing the primary metric but can do a better job on optimizing the secondary metric. This may also explain why the benefits due to INT are relatively modest, since it only wins on the secondary metric.

Exploring finding (2) leads to the following underlying results:

2a **Different INT signals result in different tradeoffs between performance metrics.** We find that within the extreme scenarios from above, queue-based INT metrics result in CCAs that maintain high throughput for a tradeoff of extra delay, while link-based metrics (*e.g.,* link utilization) result in very low delay in many cases.

2b **While different INT signals perform similarly on average, there are specific scenarios where an INT signal dominates.** Specifically, in these extreme scenarios, we find that queue-based INT signals shine in scenarios with very few senders. Meanwhile, link-based INT signals help in scenarios with long flows and many senders, but do not do well with very high numbers of senders.

We show that these results are robust across several different network topologies, traffic matrices, and specific forms of signals. Of course, the scope of our evaluation is not exhaustive or representative of all valid use cases. Thus, we encourage the application of our evaluation framework to other scenarios and hope to explore the same in future work.

2 Approach

This paper presents a large collection of simulation results, but they are all tied together by a single intellectual approach, which we now describe.

To quantify the performance of a CCA we must specify three important factors under which it operates. The first is the range of network characteristics (*e.g.,* link rates, delays, buffer sizes) and workloads (*e.g.,* number and nature of flows) under which the CCA is expected to operate. We refer to this as the range of network **scenarios**, which we denote by a set N. Second, we must specify what INT and end-to-end **signals** are available, which we describe by a set S. Finally, we must specify what performance objective is used to evaluate the CCA. Following the NUM paradigm [22], we define our objective as the average of a per-flow **utility function** u, denoted P_u.

For a given network scenario N and utility function P_u, we are interested in what in-network telemetry signals S a network should offer. Hence, we do not focus on what a specific CCA would do with such signals but rather what is the best *any* CCA could do. If we could determine the optimal CCA for any given N, S, and P_u, then we could address the above question by simply evaluating the optimal CCA for different candidate S. For example, we could simulate the optimal CCA for each S over different network scenarios in N and pick the S that yields the maximum P_u. However, there is currently no known solution that searches the CCA design space to find the optimal CCA and hence we must resort to practical approximations of the optimal.

As discussed in the previous section, our strategy will be to leverage learning-based

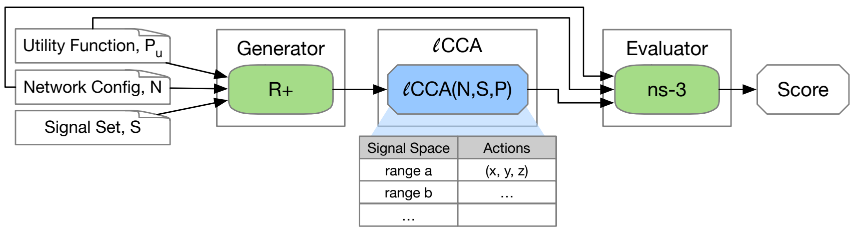


Figure 1 An overview of our signal evaluation framework. The network configuration, the utility function, and the signal set are configurable parameters for R+ while the score and lCCA(N, S, P) are artifacts produced by the process.

approaches to generate a CCA that we use as our approximation of the optimal. To that end, we define lCCA(N, S, P_u) to be the learned CCA that, given the set of signals S, optimizes the performance objective P_u when averaged over a set of network scenarios N. Thus, we are interested in, for a given range of networks N and performance objective P_u, how well lCCA(N, S, P_u) does as S varies.

With lCCA as our approximation of an optimal CCA, we now provide an overview of our evaluation process, leaving a detailed description to §4. Our evaluation uses a learning-based CCA generator and a network simulator. In our work, we used an improved version of Remy [42], which we call R+, to generate our learned CCAs, and ns-3 [2] as our network simulator to evaluate lCCAs. We believe our process could easily be extended to other CCA generators and simulators.

The process of evaluating a signal set S consists of three main steps as shown in Figure 1:
(1) Generating an lCCA using R+. First, we provide R+ with a network configuration representing the range of network scenarios the CCA should consider (N) and the set of signals it is able to consider (S). We also set the utility function that R+ uses to evaluate and optimize candidate algorithms (P_u). Given these parameters, R+ generates lCCA(N, S, P_u).
(2) Validating lCCA using simulation. CCA generator tools such as Remy typically rely on a high-level network model to evaluate the algorithms and rules they generate. Network simulators such as ns-3 typically offer greater realism since they can easily create complex topologies and model lower level effects such as the details of TCP sockets. Thus, as an additional validation step, we take the lCCA generated in the previous step and evaluate it in ns-3 [2], recording the overall performance P_u (or "score") that the lCCA achieves.
(3) Picking lCCA. We repeat the above steps for multiple checkpoints of training and finally select the lCCA that receives the best score (*i.e.,* best P_u) in simulation as our approximation of the optimal CCA for a given N, S, and P_u.

We repeat the above steps for different sets of S (including no in-network signals) and compare their relative performance to pick the best S. Thus, for a given utility and network scenario, we can determine the best utility score that can be achieved for different signals, and hence which signals were the most useful. We conduct this process for multiple utilities (P_u) and different network scenarios (N). Of course, there are an unbounded number of possible configurations, so we only pick a reasonable initial set (§4).

3 Finding lCCA

To approximate the optimal CCA, we turn to prior work in automating the development of congestion control. While there are many works in the space of automated CCA development that may be applicable ([45, 3, 20] for a sampling), we build on Remy [42]. In this section, we start with a brief overview of Remy and how we extended it to create R+, and then discuss our rationale for building on Remy.

3.1 Remy and R+

We review baseline Remy [42] using the notation from §2. Remy trains CCAs to maximize performance according to a utility function, P_u. Given a range of network configuration parameters N (*e.g.,* bottleneck rate, RTT) from which to sample, Remy produces a CCA in the form of a match-action table of "rules". The rules in the generated CCA cover a space of signal values and map to an action for the CCA to take; *e.g.,* change the cwnd and the pacing rate based on observed RTT value. Remy has a notion of *generations* that guide the degree of optimization that the tool attempts. In each generation, Remy picks a random sample of networks within the parameters of a provided configuration. Then, starting with the rules from the previous generation, Remy intelligently steps through each rule, testing variants of a rule by evaluating whether they lead to better performance as measured by P_u. We refer the reader to [42] for a full description of baseline Remy.

For our work, we made a few small, but important, modifications to Remy. These primarily involved increasing the network configuration sample size (which we found was necessary for consistent results), adding support for in-network signals, and extending the action types that Remy considers. The latter was particularly important to achieve better performance. Specifically, we introduced actions that allow the pacing rate at the sender to be a *function* of other observed performance metrics, rather than being directly set. We discuss these changes in more detail in Appendix A.1. Our evaluation showed that these changes improved the performance of our lCCAs by up to 3x in some evaluation scenarios (A.1). To avoid confusion, we use R+ to refer to Remy with our extensions.

3.2 Why Remy (and R+)?

We chose to build on Remy because it has a few desirable properties: (i) its capabilities and limitations are well-studied [39, 44, 3], (ii) it produces interpretable artifacts, and (iii) it could be easily extended/improved for our use case, as we did with R+. We acknowledge that even with its improvements over the original, Remy (or R+) may not be the ideal search algorithm for finding optimal congestion control algorithms, but we believe it to be the best prior work available which produces an interpretable CCA optimized for average-case behavior. Importantly, prior work has shown that Remy serves as a "tractable approximation of the optimal" for a given utility function [44] – the exact property we need. Further, since it produces interpretable artifacts (*i.e.,* not online or deep-learning based), it is more amenable to refining its performance, auditing its outputs, and building it into a larger system.

For due diligence, we repeat and extend the evaluation of prior work, now comparing the performance of the lCCAs generated by R+ (not Remy) to the performance obtained by parameter-tuning popular CCAs (TCP Vegas, Cubic, BBR) and a deep reinforcement learning (DRL)-based CCA [20]. Our experiments showed that R+ consistently matches or outperforms the best performance obtained by parameter-tuning TCP Vegas, Cubic, and BBR. Similarly, our experiments with a DRL CCA [20] found that it produced worse-performing algorithms than our R+ outputs (A.2).

Perhaps most importantly, we repeated a sample of our experiments from §5 using the best DRL-based CCAs generated by the above and found that the high-level conclusions and findings that we report with R+ remained the same; *i.e.,* although the absolute performance values differ, the overall trends (or lack thereof) remain unchanged. Thus we believe that our findings based on R+ are representative of a high-performing CCA.

There are many known limitations to Remy [3, 39], including its ability to generalize to scenarios outside of its training configuration. Fortunately, these limitations are not of concern for our use case since we only care to evaluate how well a CCA uses a signal set S for a specific network scenario N, rather than how generalizable the CCA is for a different network scenario N'. We address these limitations and other related work further in §8.

Finally, we note that our overall methodology is agnostic to how lCCA is generated. As better options emerge for producing lCCA, or requirements change (*e.g.,* no requirement for interpretability), we hope our work provides a blueprint for how these options can be used to answer fundamental questions such as evaluating signals.

4 Evaluation Setup

We present the experimental setup for the three main factors in our evaluation: the range of network scenarios (N), utilities (P_u), and signals (S) that we consider. In each case, we discuss the parameter space in general and then present the specific parameter values we experimented with. Finally, we elaborate on relevant aspects of how we configure R+ and ns-3 for generating and evaluating lCCAs. Notably, for N, P_u, S, and other parts of the experimental setup, we attempt to select a reasonable initial value. However, we hope that our framework will be used to explore more configurations, topologies, *etc.* in future work.

4.1 Network Scenarios

We primarily trained (and evaluated) on network configurations pulled from [42] as a starting point that reproduces prior work. Our only modifications were to change their unit of time to reflect a datacenter environment, and adapting certain parameters (*e.g.,* RTT, rate) to have a range of values instead of constants. The parameters that describe a network scenario are shown in the second column of Table 1. We see that these parameters characterize both performance characteristics (*e.g.,* RTT, link rate, link loss, buffer size) and the traffic workload (*e.g.,* via number of senders, and their on and off times).

Parameter settings: The exact parameters used for evaluation are sampled from the space defined by the configurations in Table 1. For each parameter, there is a min value, a max value, and a "step" size which determines the granularity of the values between the min and max. When sampling specific network configurations (or "scenarios") from the above, a value for each parameter is randomly selected within the set of values between the min and max value at the granularity of the step size. Therefore, the number of potential network scenarios in one of these configurations can be very large ($\sim$ 1.75 million for the default config). [1] Our results in §5 are generated/evaluated with the default network configuration in Table 1 unless stated otherwise.

4.2 Utilities

We used the utility function used in [42] due to its fairness properties (proportional throughput and delay fairness). This utility function is shown in Equation 1. Our only modification is to

[1] We note that R+ uses a unitless value to measure time. While we map this to μs by default, there is no fundamental reason why this cannot be set to any other value. Instead, it is the relative value of the parameters based on time that matters.

Config	Parameter	Min	Max	Step
default	rtt (μs)	20	30	1
	rate (pkt/μs)	2	4	0.5
	senders	1	16	1
	buffer size (pkt)	100	250	25
	off time (μs)	100	1500	100
	on time (μs)	1000	10000	500
	loss (per pkt)	0.001	0.001	0
high senders	rtt (μs)	10	20	1
	rate (pkt/μs)	2	4	0.5
	senders	16	64	2
	buffer size (pkts)	1000	1000	0
	off time (μs)	1000	1000	0
	on length (pkts)	20	10000	100
	loss (per pkt)	$1 * 10^{-6}$	$1 * 10^{-6}$	0

Table 1 Network configurations used to generate and evaluate lCCAs

add the weights c_t and c_d to the throughput and delay components.

$$u = c_t * log_2(throughput) - c_d * log_2(\overline{delay}) \tag{1}$$

Note that, in the above equation, throughput is normalized by the maximum fair-share throughput a flow can achieve. Similarly, the delay is normalized to ensure both the throughput and delay values are similar in magnitude [42]. Our added coefficients, c_t and c_d, allow us to generalize this utility function to express different goals. We set these to represent relative prioritization between throughput and per-packet delay.
Parameter settings: In §5, we report results for five utility functions of this form:

- **Default**: $c_t = c_d = 1$
- **2t**: $c_t = 2, c_d = 1$
- **2d**: $c_d = 2, c_t = 1$
- **100t**: $c_t = 100, c_d = 1$
- **100d**: $c_t = 1, c_d = 100$

Qualitatively, we will refer to 2d and 100d utilities as delay-focused and the 2t and 100t utilities as throughput-focused. When comparing CCA's achieved utility, we will refer to it as their *score*. Finally, we note that our methodology easily accommodates different utility functions, so we are not fundamentally limited to the above utilities. We leave the exploration of additional utilities to future work.

4.3 In-Network Signals

We collect two (base) forms of in-network signals, queue and link data, and try some variations on these signals. Our baseline queue metric is the queue length at a given hop, and the link metric is the rolling average utilization of the link in discrete intervals of time. By default, the time interval for collecting link metrics is $10\mu s$. We perform a sensitivity analysis of the collection interval and investigate additional signals in §6.3.

When collecting in-network signals at each hop, a natural next question is what aggregation function to apply along the path (*i.e.,* min/max/avg). With one hop, all aggregation functions are the same. For general topologies, we assume the min or max of the signal along the path, picking whichever is appropriate for the signal in question: *e.g.,* maximum queue length or maximum link utilization at any hop.

To gather in-network signals, data is collected as a packet is sent to the wire and added to a custom packet header. For example, in our simulations, an 8-byte header containing two 32-bit fields for link and queue metrics is inserted between the L2 and IP headers. At each network hop, this header is updated with the appropriate value (*e.g.,* maximum queue length) seen along the path.

Parameter settings: In our evaluation, each CCA has a default set of end-to-end (E2E) signals available to it. We use the baseline signals in Remy as our default set, which includes: an exponentially-weighted moving average (EWMA) of the sending rate and the receiving rate, a "slow" EWMA of the receiving rate with a lower weight for new data points, and the ratio of the latest RTT measurement to the min RTT measurement seen so far. Unless otherwise specified, all CCAs have access to these default signals. Notably, this default set does not include loss – we will explore the role of loss signals in §5.1.

Our evaluation considers *augmenting* the above default E2E signals by in-network queue and/or link signals. Thus, we have 4 essential options for evaluation:

- **E2E**: only the baseline end-to-end signals
- **Link**: E2E signals + link telemetry signals
- **Queue**: E2E signals + queue telemetry signals
- **Link + Queue**: E2E + link and queue telemetry signals

Note that we do not explicitly model ECN since this is subsumed by our queue length signal. In general, to answer questions about signal quality, we will compare the relative performance of CCAs trained with the above signals sets.

4.4 R+ Setup

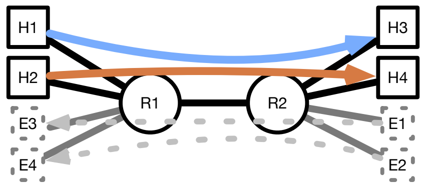
In generating lCCAs, we run R+ for five training generations, as we found that performance typically stops improving after this point. The current lCCA is recorded after each generation.

In training, the candidate lCCAs are evaluated repeatedly on a set of 200 simulated network scenarios. The network model in Remy, parameterized by the variables in Table 1, is necessarily simple to allow for tractable evaluation of many candidate algorithms. In this model, all senders reside on a single host directly connected to the receiver. Accordingly, all flows compete on exactly one shared link. The parameters of this link are defined by the particular sampled network scenario: rate, delay, buffer size, and stochastic loss rate (buffer overflow will also result in loss but is not configurable). Similarly, the specified number of senders turn off/on for durations determined by sampling from an exponential distribution with the scenario-specified mean.

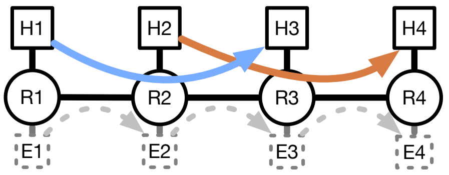
4.5 ns-3 Setup

Once R+ has completed training, the lCCAs from each generation are evaluated in the ns-3 simulator. This evaluation is run on 200 network configurations randomly selected from the original training set, with 10 simulations per configuration to ensure reliable results. The CCA with the highest score is selected as our final lCCA for the given (N, S, P_u).

By default, we simulate a topology similar to the model used in training: a single-bottleneck dumbbell topology, with the host link rates and latencies set to have no significant impact on the results (Figure 2a). All other parameters match the network configuration sampled from the space defined for training (*e.g.,* bottleneck rate, latency, senders, etc.).



(a) Dumbbell topology with 2 senders and optional reverse path flows.



(b) Line topology with 1 host / router, 2 senders, and optional extra flows.

Figure 2 Topologies used to evaluate the produced CCAs. All host links are set to 1 Tbps rate, 0 ms latency, and 10,000 pkt buffers to avoid influencing results. Flows are designated with arrows (dotted for extra flows to create multiple bottlenecks or reverse path congestion).

To test how well the results generalize to other topologies, we also evaluate on a "line topology" as pictured in Figure 2b with each sender on one half of the topology sending to one receiver on the other side. As with the dumbbell, all parameters are set to match the configuration of the given network scenario. We chose this topology to serve as a generalization of the dumbbell topology while remaining true to the network configurations with which each CCA has trained. We can vary the number of hosts connected to each router between the base case of a dumbbell topology and the other extreme with only one host connected to each router.

While these two topologies are quite simple, we use the line topology and additional flows to capture important variations over the training scenario: multiple hops, different bottlenecks between competing flows, and reverse-path congestion. On the line topology, we can optionally enable additional flows to test our CCAs under multiple bottlenecks. In this version, there is an additional flow crossing each individual router-router link which only intersects the other traffic for one hop. This allows each sender to have a potentially different bottleneck, depending on which of the extra senders are running at a given time (as shown to be an important use of INT in [41]). Similarly, as shown in Figure 2a, reverse-path senders can be enabled on the dumbbell topology.

Finally, we note that accurately simulating lCCAs required making a few modifications to ns-3 to enable correct timestamping and fine-grained pacing at the sender. These modifications are described in more detail in §A.1.

5 Results

We now present the results from using our framework to analyze the value of INT signals. The results show that for average case utility (score), the value is minimal. However, analyzing individual network scenarios and other performance metrics reveals the merits of INT and the specific strengths of each signal type.

Figure 3 shows the utility score averaged across all evaluated network scenarios, for different signal set options. The error bars indicate the standard deviation in scores which we found to be small ($\leq 5\%$). We observed a similar trend for our 100t and 100d utilities and hence omit them from the figure due to space considerations.

We see two trends from Figure 3: (1) for a given utility, there is no significant difference between CCAs that use INT signals and those that use only E2E signals, and (2) for a given utility, the average score varies very little across different INT signal sets (Link, Queue, Link + Queue). We found these results surprising and hence investigate each in §5.1 and §5.2 respectively.

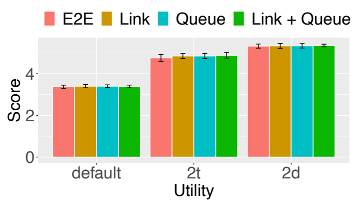


Figure 3 Utility score averaged across all network scenarios evaluated for default, 2d, and 2t utilities and all main signal sets.

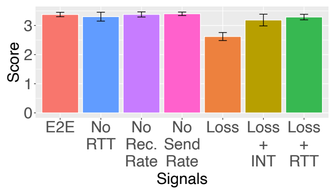


Figure 4 Default utility averaged across all network scenarios for signal sets with fewer signals than the baseline (E2E).

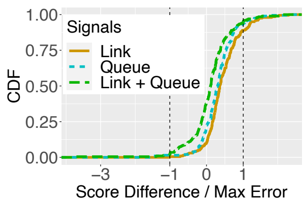


Figure 5 Distribution of score difference significance, $s(score)$, between each singal set and E2E for default utility.

5.1 INT and End-to-End Signals

Early work on leveraging ECN demonstrated substantial performance gains over loss-based CCAs [34, 5, 16, 15, 43] and hence it is (arguably) somewhat surprising that INT-based CCAs do not generally dominate E2E ones in Figure 3. Recall, however, that our E2E CCA in this figure uses a richer set of E2E signals - RTT, send rate, and receive rate. Hence, one hypothesis is that one or more of these richer signals enable better performance than traditional loss signals, effectively “closing the gap” between E2E and INT-based CCAs. To test this conjecture, we repeated our evaluation with different options for E2E signals, including systematically removing one signal at a time from our default E2E set, and considering loss as a signal both individually and with other signals. Our results are shown in Figure 4.

We see that the scores remained essentially the same even as we removed specific signals from our default E2E set. This is shown with the bars labeled “No RTT”, “No Receive Rate” and “No Send Rate” in Figure 4, each of which removes the named signal from the E2E signal set. We see that these trained CCAs – despite having access to only two of the E2E signals – were able to reach the same overall utility.[2]

Figure 4 also shows that a CCA that uses only packet loss as its signal (like NewReno, Cubic) produced significantly lower utility. In contrast, an algorithm that (in addition to loss) is given access to RTT or INT signals was able to achieve performance comparable to those using a full set of signals. This suggests that *when measured by average performance, nearly any set of signals can be effective, provided they are more proactive than packet loss alone* and the algorithm’s control loop is optimized for them. To better understand this result, we further breakdown CCA performance in the rest of this section and find that, while this is true in the common case, INT signals can provide value in some network scenarios.

From Average to Specific Network Scenarios All signal sets may seem to perform similarly due to the effect of averaging across all network scenarios within the considered space (defined in Table 1). While two CCAs may have similar *average* performance, there may be specific network scenarios in which the difference in performance is significant (*i.e.,* larger than the error for the given network).

[2] We found that training the version with no RTT signal, required a minor change to the exact form of the sending and receiving rate signals. In this case, we normalized all rates to the receiving rate so that Remy did not have to create a rule for all possible relationships between sending and receiving rate. We also required 8 generations of training (vs. 5 earlier) for the score to stabilize. We experimented with this modification for other signal sets, but saw no significant change in utility.

In order to analyze these differences more closely, we look at the full distribution of performance across all network scenarios evaluated. To determine the significance of any difference, we normalize to the maximum error of the two CCAs being compared. More precisely, we calculate $s(p)$, our significance value for performance metric p between signals sets a and b, as

$$s_i^{a,b}(p) = \frac{p_i^a - p_i^b}{max(e(p_i^a),\ e(p_i^b))} \tag{2}$$

where p_i^x and $e(p_i^x)$ are the performance metric and its error for the signal set x on network scenario i, respectively. We plot the distribution of $s(score)$ in Figure 5 for the default utility, showing $s(score)$ for each INT signal option computed relative to our E2E signals. I.e., in Figure 5, a from Eqn. 2 is one of Link, Queue, or Link+Queue, and b is the E2E signal set. We consider a score difference s_i insignificant if it lies in the range (-1,1).

As shown in the figure, for the majority of network scenarios, the performance difference between having any INT signal and none is insignificant (*i.e.,* $-1 < s_i < 1$). However, there are up to ∼11% of network scenarios where the performance gap is significant for the default utility function.

Figure 6 extends this analysis to our other utility functions. We plot the distribution of s_i between the Link + Queue and E2E CCAs for all utilities. We observed similar results with Link and Queue individually but omit them for clarity. As before, we observe that for many utilities the number of scenarios with significant differences is relatively low (*e.g.,* 2d, 100t) but we also see that large differences can be common for some extreme utilities (*e.g.,* 100d). Closer examination reveals that while the difference in scores is large relative to the error (denominator in Eqn. 2), this difference is often small relative to the score itself. We see this in Figure 7 which plots the distribution of the same score differences but now normalized to the E2E score (vs. the error). We see that, in almost all utilities, no difference is greater than 10% of the E2E score (100t is the only exception). Thus, even though there may be a large number of significant differences, these differences can be small enough that the average remains within average error (as seen in Figure 3).

Our results above align with prior work on INT-based CCAs which demonstrates that INT signals are particularly helpful in especially difficult scenarios; *e.g.,* to disambiguate bottleneck delay from end-to-end delay when there are multiple bottlenecks [41]. Given the extensive research on optimizing performance in extreme network conditions [28, 37, 5, 24, 7], understanding these differences can be particularly important for network operators supporting demanding applications. In addition, operators may want to understand the implications of operating in a specific scenario (*e.g.,* level of demand, buffer size). Thus, we aim to determine both *how* these signals can improve performance and *in what situations* they are most valuable.

***How*: Breaking Down Utility** Further analysis of the individual components of the overall score (throughput and delay) revealed more about why these results arise. Figure 8 shows the percentage of network scenarios for which the signal set outperforms the baseline in the "primary" and "secondary" metrics (*i.e.,* $s(tput) > 1$ and $s(delay) < -1$ where the primary metric is the one with the larger coefficient in the utility function). As shown in the figure, INT signal sets generally outperform the E2E signal set in the secondary metric, rather than the primary. And, this difference is larger in the utilities with the most significant score differences (*e.g.,* 2t and 100d in Figure 6). These results suggest that *INT is better at*

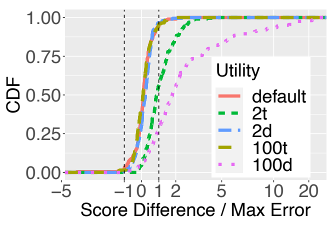


Figure 6 Distribution of score difference significance between the Link + Queue and E2E signal sets for all utilities.

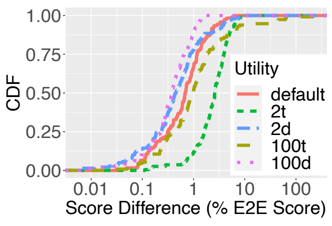


Figure 7 Distribution of score differences between the Link + Queue and E2E signal sets normalized to E2E score for all utilities.

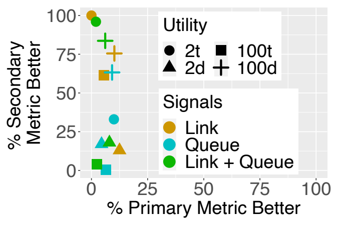
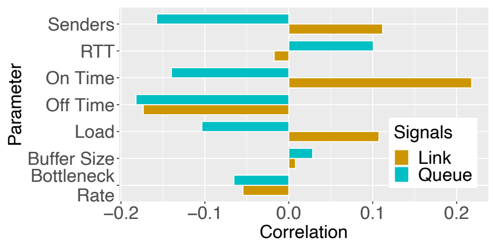
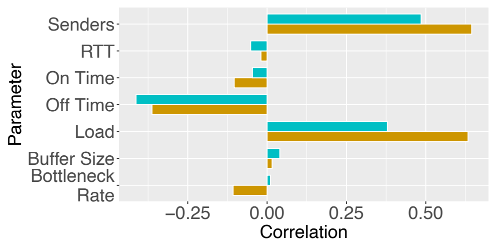


Figure 8 Percent of network scenarios with significantly better primary metrics compared to E2E for all non-default utilities.



(a) Default utility.



(b) 2t utility.

Figure 9 Correlation between $s(score)$ and network parameters. A large positive (negative) value indicates the that the given signal set tended to outperform the E2E CCA as the parameter increases (decreases). Smaller magnitude correlations indicate that the network parameter was not a significant factor in the score differences.

balancing the primary and secondary metrics than E2E which may often have to sacrifice the secondary metric in order to maximize its utility overall.

***When* INT Matters** To understand the differences shown in Figure 6 more deeply, we look at which network scenarios result in significant performance gaps. Specifically, we measure the correlation of the score significance, $s(score)$, with the parameters of the evaluated network scenario. The calculated values for Link and Queue are shown in Figure 9a for default utility and Figure 9b provides an example for non-default utility. As an example, Figure 9a indicates that Queue tended to significantly outperform the E2E signals when there were few senders (relatively large negative correlation between $s(score)$ and the number of senders), while Link did better with more senders.

Using this analysis across utilities, we found some consistent patterns. INT signal sets helped with throughput-focused utilities when the off time for flows was low (*i.e.,* the number of flows is usually close to maximum). For delay-focused utilities, INT helped with low load (*i.e.,* senders per unit bandwidth) and low numbers of senders. These cases reflect the strengths in Figure 8 as INT helped achieve higher throughput in less-loaded scenarios for delay-focused utilities and helped with delay for throughput-focused utilities by handling the instability of many flows.

This appears to be the benefit of the “global” nature of INT: *i.e.,* an INT signal captures router or link state in its entirety while traditional E2E signals like RTT measurements can only reflect the experience of a specific flow at a router/link. As an example, under high load, if a flow arrives during a period of congestion, it may measure an inflated RTT

Utility	Better Score	
	Link	*Queue*
default	4.5%	0.5%
2t	0%	1.5%
2d	0.5%	1.5%
100t	3%	6.5%
100d	28.5%	8.5%

Table 2 Percentage of network scenarios with significant score differences between Link and Queue CCAs.

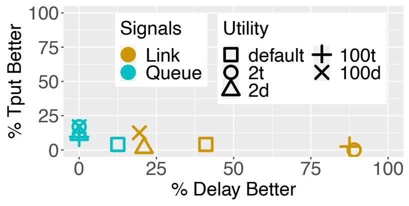


Figure 10 Percentage of significant throughput and delay differences ($|s(p)| > 1$) between Link and Queue CCAs for all utilities.

for its minimum RTT. A CCA with a queue metric, however, will be able to tell that this min RTT measurement is not the true minimum. Similarly, a low-load scenario may be hard to distinguish from a higher-contention setup in a good steady state without more global information (*i.e.,* the RTT may be close to the min RTT in either case, though the link may be underutilized under low load). More global knowledge like this may help the INT-based CCAs generalize better across loads (as the gap between local information and global widens).

As shown in Figure 9, however, the two individual INT signals sometimes have different strengths (*e.g.,* number of senders). We explore these differences next in §5.2.

In sum, while the E2E CCAs achieve the same utility in the average case as all other signal sets, there are many scenarios in which their limitations result in worse performance particularly for non-prioritized metrics. Further, as we discuss later and in the Appendix, E2E CCAs generalize to some scenarios poorly (§6) and often have the worst fairness.

5.2 Analysis of INT Signals

We now analyze each individual INT signal in order to determine why they perform similarly in the average case. This analysis takes similar form to the comparison of E2E and INT: we consider both the individual performance metrics and individual network scenarios that produce significant differences.

Finding Differences Table 2 summarizes the number of significant utility differences between Link and Queue signals ($|s^{Link,\ Queue}(score)| > 1$) for all utilities. Note that we are now comparing one INT signal against another (versus comparing against E2E as we were doing in the previous subsection). As seen in the table, there are some significant differences, especially in the more extreme utilities. As before, these differences, while significant, are generally small – *i.e.,* less than 10% of either CCA's score (except for 100d which has differences up to ∼20%). Thus, as with E2E, the differences are both infrequent enough and small enough to average out.

***How*: Throughput / Delay Tradeoffs** We now look at the individual components of our utility functions to determine if each INT signal achieves its overall utility differently. The percentage of network scenarios for which $|s(p)| > 1$ between Link and Queue across utilities is shown in Figure 10. The Link CCAs generally achieve better delay in more scenarios, while Queue CCAs achieve better throughput in more scenarios (though the percentage is much smaller). Further, Link outperforms most in delay when throughput is the primary metric (2t and 100t), but the gap reduces when both versions are forced to prioritize delay

(2d and 100d). In the score distributions, these differences often do not amount to much of a difference, especially when the metric is not emphasized (*e.g.,* Link outperforms Queue in ∼88% of the delay distribution when the utility function is 100t but this has minimal-to-no impact on the overall score).

***When*: Strengths by Scenario** We now consider the network scenarios in which one signal performs better than another. With non-default utilities, Link and Queue tend to help in the same kind of network scenarios (as in Figure 9b) indicating that these scenarios are more about the weakness of E2E than the individual signals. Under default utility (Figure 9a), however, there are some notable differences. For example, Link does well with long lived flows (correlated positively with on time) while Queue does worse. Similarly, we see that Queue's performance is negatively correlated with the number of senders while Link is positively. More concretely, we found that in the top 10% of senders, Link had a significantly better score than E2E in 15% of cases (vs. 8% for Queue), while Queue had a better score in 27% of cases in the bottom 10% of senders (vs. 0% for Link). In both extremes, the signal set with better delay is the signal set that performs better overall (the throughput differences are generally more insignificant).

Intuition Without a queue metric to accompany it, a CCA with only link metrics is unable to accurately perceive the *degree of overload* it may be causing. Thus, it must keep the link utilization near but below 100% in order to avoid queueing that may impact the score if large enough. This may lead to the tradeoff we see where Link achieves better delay but worse throughput than Queue in many cases. However, as mentioned above, under low load and default utility, Link often has significantly worse delay than Queue, indicating that it may adopt a different strategy in these regimes that sacrifices some extra delay for better throughput (to avoid severe underutilization with low load).

As we will explore in §6.2, in settings with significantly more senders, this naive approach of trying to under-utilize the bottleneck to avoid delay can work poorly.

Queue metrics, on the other hand, allow the sender to ensure that the bottleneck link is fully utilized (queue > 0), but are unable to perceive the *extent* of under-utilization. Hence, they tend to maintain a queue (and therefore some delay) in order to achieve higher utilization, but are able to bound this queueing delay caused by high utilization better than CCAs that do not have access to queue information. Further, this approach generalizes well to low load: senders should increase rate until a small queue is built instead of leaving throughput on the table (or having to significantly sacrifice delay in order to achieve high throughput).

Both Link and Queue Signals Naturally, as we have seen link and queue INT to provide different strengths, having access to both generally lands somewhere in between. Again, this is more apparent with more extreme utilities. Inspection of the full distribution shows that for many network scenarios the performance gaps, particularly for the non-prioritized metric, are significant. For example, the Link + Queue CCA achieves significantly better delay than Queue in 91% of scenarios for 2t utility, while it achieves significantly better throughput than Link in 11% of scenarios for 2d utility (with an insignificant difference in 88%). As before, these differences do not always amount to a significant utility difference: *e.g.,* Link + Queue only achieves a significant difference over Queue in 7% of 2t utility scenarios despite the large number of significant delay differences. Lastly, we find that Link + Queue CCAs

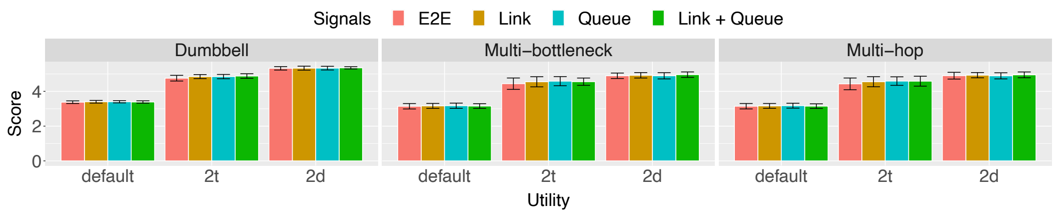


Figure 11 Average scores for different signal sets and utility functions across different topologies.

generally achieve significant utility differences over E2E in similar scenarios to when Link or Queue perform well.

Summary In summary, different signals navigate the trade-off between maximizing throughput and minimizing delay in distinct ways. Algorithms using link metrics excel at minimizing delay by keeping the link just below 100% utilization, a strategy particularly effective with a sufficient load/number of senders. Conversely, algorithms using queue metrics are better at maximizing throughput by maintaining a small queue to ensure the link is fully utilized, which is advantageous in low-load conditions. Combining both link and queue signals provides the best balance, achieving better throughput than link-only versions and better delay than queue-only versions.

Each of these results presented was based on our default network configuration. Thus, we consider a different network configuration in §6.2. Similarly, we address several variants of INT signals in §6.3. See the Appendices for details on methodology and a discussion of issues beyond end-to-end performance.

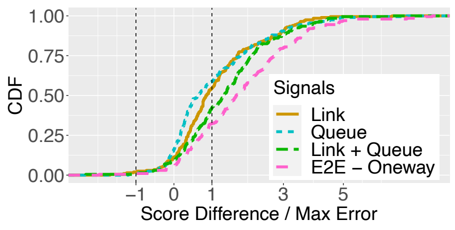
6 Robustness

6.1 Topological Variations

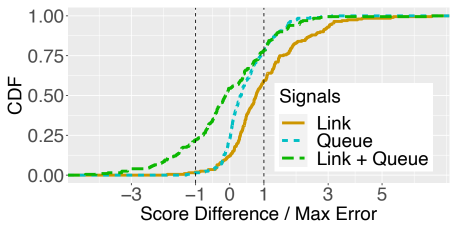
Recall that our first result was that *on average, over all network scenarios,* all sets of signals performed equivalently. We now explore if this result applies to a wider range of network topologies and competing traffic.

Multi-hop Topologies We evaluated the CCAs (trained on the single-bottleneck network model) on the line topology shown in Figure 2b with one host connected to each switch. We find that, as with the original topology, there is no significant difference in the overall utility. As shown in Figure 11, the average scores are within error of one another, though lower than they are on the dumbbell topology. Closer inspection of the distribution of score differences revealed few significant differences between the CCAs with INT vs. E2E signals (especially as the error for all signals, but notably E2E, increased). Thus, our basic result holds for these multi-hop topologies.

Multi-bottleneck Topologies Similarly, we evaluate our generated CCAs on a multiple hop topology with additional senders placed in order to have different bottlenecks for different flows (as shown in Figure 11). Here, we again see that the overall takeaways of previous results remain the same.



(a) One-way signals.



(b) Two-way signals.

Figure 12 Distribution of significant score differences against baseline E2E signals (which are two-way by default) for all signal sets with reverse path congestion.

Reverse Path Congestion Lastly, we consider an alternate setup with congestion on the reverse path of our evaluated CCAs (as shown in Figure 2a). Here, we set the reverse path to have twice the number of senders and generate sustained congestion.

There is an additional relevant and potentially significant difference in this setup: one-way signals. In our baseline setup, the INT signal is collected at every hop along the entire two-way path; *i.e.,* forward and reverse path. An alernate option is one-way INT signals as discussed in [35]. Here, the signals are "locked" at the receiver and reported to back to the sender, without editing on the reverse path. In our experiments so far, since there was no reverse-path traffic, the difference between one-way and two-way signals was insignificant. Now we evaluate the potential impact of reverse-path traffic.

The results with locked INT signals are shown in Figure 12a. Notably, INT signals perform significantly better in many cases in this regime. However, this may only be due to the fact that the signals are unidirectional and not the inherent value of the particular signals. Accordingly, we also test the E2E signal set with access to one-way delay.[3] As shown in Figure 12a, this performs slightly better than the (one-way) INT-based signals and reveals that having one-way signals may be valuable, regardless of the exact signal. More generally, this finding suggests that some of the value of INT signals may be from features like unidirectionality (or sub-RTT signaling as in [7]) rather than the metric's unique importance.

We now consider a case where all signals are two-way. With two-way signals, INT signals are now updated on the reverse path. As shown in Figure 12b, this setup results in an increased number of scenarios with significant score differences relative to the baseline evaluations with no reverse path congestion. We attribute this increase to the cumulative nature of delay. While INT signals record a non-cumulative, min/max per-hop value, the E2E delay signal is cumulative. Therefore, delay from congestion on the reverse path will compound, potentially making the overall congestion seem worse to an E2E algorithm than to an INT-based one. However, despite these cases, the INT signal sets once again show mostly similar *average* scores to the E2E signal set.

6.2 Other Network Configurations

We also trained and evaluated different signal sets on the "high senders" network configuration shown in Table 1. The main differences between this configuration and the default configuration are the number of competing senders and their on-off pattern. In this configuration, we

[3] This is, of course, a non-trivial signal to achieve in a real deployment. We use this test in simulation to evaluate whether, if possible, a one-way version of E2E signals could achieve the same performance as one-way INT signals.

Metric	Link + Queue		Link		Queue	
	Better	*Worse*	*Better*	*Worse*	*Better*	*Worse*
Utility	32.5%	9%	21.5%	39%	59%	0%
Tput	82.5%	0%	77.5%	0%	72%	0%
Delay	0.5%	84.5%	0%	96.5%	3%	78%

Table 3 Comparison of different INT signals against the E2E set for the "high senders" config (Table 1). Each cell contains the percent of scenarios in which the given signal performed significantly better or worse than E2E ($|s(p)| > 1$).

use "byte-switched" senders which have a certain number of packets they must send each time they turn on rather than an amount of time during which they must send data. This ensures that flow sizes are not a function of the available bandwidth. More importantly, the number of senders has 4x the maximum value of the default config.

As with the default network configuration, all overall score averages are within error of one another – *i.e.,* INT does not provide a score difference on average, and all INT signals are similar on average. However, as shown in Table 3, Link achieves worse delay for similar throughput to all other INT versions. All INT versions generally see worse delay than E2E (as it sacrifices throughput to keep delay low), though Link is the worst. Notably, this is in contrast to its previous strength against Queue (§5.2).

Upon further inspection of results, the Link version has a significantly worse score than the other versions when there is a high number of senders: *e.g.,* Link has a significantly worse score than E2E in 75% of scenarios in the top 10% of senders. While the Queue version does sometimes get less throughput than the Link CCA, it is able to effectively mitigate the delay, even in high contention scenarios. This result demonstrates that sensitivity to the degree of congestion can be critical. In the default setup, naively running close but under 100% utilization was an effective strategy, but with the instability of more senders, the Link CCA cannot as effectively prevent queueing delay. By having access to both metrics, the Link + Queue CCA is able to better balance the two metrics.

6.3 INT Variations

We now analyze a few variations of the INT signals to determine the impact (if any) on the relative value of the signals. These variations are important as switch vendors may need to ensure that the metrics they expose are the most useful form for CCA developers.

Available Capacity vs. Utilization While we use percent utilization as our link signal by default, we tested another form of the link signal: available link capacity (AC) measured in bps. We chose this as an alternative since it contains more information about the actual link capacity than utilization. Therefore, a sender may be able to reason more precisely about how quickly to send data by understanding exactly how much bandwidth is available.

Yet again, in the overall score, the two options perform essentially equally well. There are however, many network scenarios in which the two versions perform significantly differently (up to $\sim 35\%$, depending on the utility). As shown in Figure 13, utilization-based link metrics generally led to better throughput, while AC generally led to better delay. Further, we found that AC versions generally perform better than utilization versions in network scenarios with high load and senders, while performing worse than the utilization versions under low load.

Intuitively, these relative strengths reflect when the signal has the most information. In low-load situations, link utilization is a more effective signal for achieving throughput

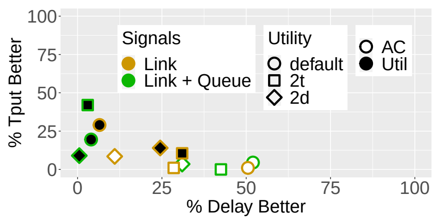


Figure 13 Percentage of significant throughput and delay differences between AC and utilization-based versions of link signals.

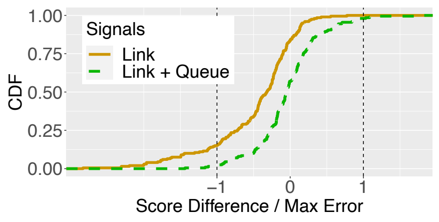


Figure 14 Significant score differences between baseline versions and 10x collection interval versions.

because it generalizes well across links with different capacities. As a relative percentage, a low utilization value is a clear instruction for a sender to increase its rate, regardless of the link's total speed. In contrast, an absolute value for available capacity is ambiguous without the context of the link's maximum capacity, or "ceiling".

Link Data Collection Interval As mentioned in §4, we set the default time interval over which utilization metrics were collected to $10\mu s$. To determine the significance of this choice, we increased the value to $100\mu s$ and trained CCAs with access to link signals. This change had no clearly significant impact on the score of either the Link or Link + Queue CCAs on average. As shown in Figure 14, the longer interval did significantly decrease the achieved utility in some cases for Link. These cases largely had low on times and high off times, indicating that the longer collection interval hurt performance when flows are short and sparse. In these cases, the longer window results effectively averages the utilization over a longer time, making the signal less accurate. Notably, this effect did not seem to matter when a Queue metric was also available (only 2% of cases had significant differences for Link + Queue). While we increased the collection interval by 10x, further increases may see more dramatic impact. We leave this to future investigations.

Smoothing We found that details of how metrics are smoothed into an INT signal matter. For example, with queue length metrics, simply running them through our default EWMA proved to be a poor choice, resulting in significantly lower average utility. The impact of a transient queue would persist for long after the queue had drained leading the CCA to be unnecessarily conservative. However, an EWMA proves to be the right choice for link metrics.

7 BBR Evaluation

In order to confirm that our finding – namely that INT signals do not provide much value in the average case – was not simply an artifact of our methodology, we modified an existing CCA to use INT signals. The selection of CCA was important since we should not change the control loop logic of the CCA (which may improve the performance regardless of the quality of the signal, and could make the CCA arguably just a different algorithm altogether). Thus, we chose BBR [10][4] since it has a clear goal state that is independent of the signals

[4] We note that we specifically used BBRv1 as a starting point since it is a simpler protocol and is readily available for ns-3.

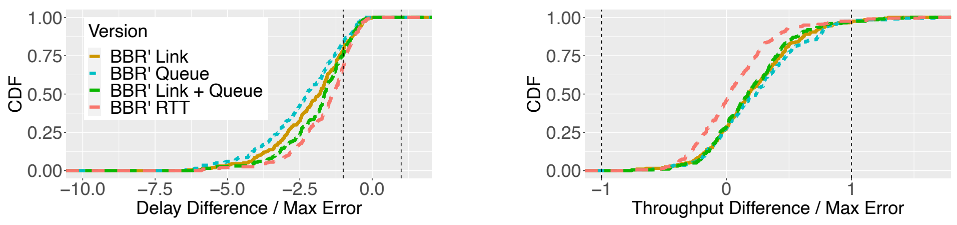


Figure 15 Significant throughput and delay differences between BBR' using different signals and baseline BBR.

used to achieve this state. However, we also performed similar experiments with DCTCP in A.3. As mentioned in [10], BBR is designed to (1) keep a BDP of packets in flight and (2) send at the bottleneck rate. At a high level, it does this by probing for bandwidth and RTT estimates, which requires purposely sending above/below the current estimate of the bottleneck rate.

BBR implements this with four states: PROBE_BW, PROBE_RTT, STARTUP, and DRAIN. It is designed to spend $\sim 98\%$ of time in the PROBE_BW phase [10]. Accordingly, we modified the conditions for changing the pacing rate in the PROBE_BW phase. In PROBE_BW, the pacing rate is set above the estimated bottleneck rate, then below, then maintains the estimated rate. The phase of the pacing rate is determined partially by time spent in the phase, as well as some exit conditions (such as incurring loss). For example, the high rate phase is exited once the minimum time has passed *and* either bytes have been lost or more than a BDP is inflight.

These checks presented a natural place to add INT signals. We *added* to the conditions to change gain phases (*i.e.,* not modifying existing conditions). Specifically, required that utilization be above 90% to exit the high-rate phase and that queueing be less than 5 packets to exit the low-rate phase as an initial implementation; we arrived at these values after some experimentation. We also tested changes to the transitions between states, but found almost all changes to have no noticeable impact. The only transition change that did result in noticeable performance changes was exiting the PROBE_BW phase, where we add conditions to exit if the queue length is greater than 0 or the link utilization is greater than 95%.

As shown in Figure 15, these changes, denoted generally as *BBR'*, resulted in essentially no change in throughput performance, but a drastic improvement in the delay achieved.[5] Although this may initially seem to indicate that the INT signals were necessary to produce this result, we made similar changes to the exit conditions based on the observed RTT instead (*i.e.,* exit high rate if the RTT crosses some threshold indicating queueing, exit low rate when it gets below a threshold, exit PROBE_BW if above some threshold). As shown in Figure 15, we are able to achieve nearly the exact same results with RTT, a traditional end-to-end signal. This indicates that the improvement was based on an algorithmic change, rather than the virtue of the signals themselves, aligning with our results that nearly "any signal will do" in the average case.

These results are also robust to more complex topologies and scenarios. The same modifications to BBR on a line topology with extra flows to create multiple bottlenecks produced largely the same behavior. All BBR' versions had significantly better delay in $\sim 50\%$ of scenarios (vs. $\sim 75\%$ on a dumbbell topology).

[5] From our evaluations, this improvement in delay comes at the cost of some fairness. Further evaluation/design may determine the correct tradeoff between the two and if fairness is a necessary concession.

Signal	Optimized For	Best Config	Delay	Tput
Link	Throughput	H=95%, L=90%	1.65	0.90
Queue	Throughput	H=10, L=0	1.53	0.93
Link	Delay	H=90%, L=85%	1.52	0.88
Queue	Delay	H=2.5, L=0	1.53	0.91

Table 4 Best performing BBR' configuration for each signal determined by the median value of the metric (normalized to best possible). H and L denote the threshold for the signal to exit the high rate and low rate phases, respectively.

We also compare these results to our findings about the relative strengths of each individual INT signal. Using these signals in our BBR modification requires fine-tuning thresholds. As we manually experimented with trying to tune these thresholds for throughput or delay, our experience mirrored the tradeoffs found in §5. The best-performing version for each signal and each metric are shown in Table 4. These results, while necessarily anecdotal from a manual optimization process and notably small in performance, reiterate the strengths of the signals: Queue can achieve high throughput with a bounded but necessary sacrifice in delay, while Link can achieve low delay, but at the cost of some throughput. Overall, the signals must sacrifice the metric they observe more directly in order to optimize the other.

Our experience with applying INT signals to BBR supports our finding that nearly any signal could be sufficient in the *average case*. Of course, we cannot guarantee that a better use of the INT signals does not exist, but our best efforts did not result in any performance gain that was unique to INT signals. Further, this demonstrates that these findings may be true even when the utility function is different than our default form and the CCA is not optimized for the given setting. However, we note that other benefits of a particular signal such as understandability may be important for a CCA designer when not using a lCCA.

8 Related Work

To the best of our knowledge, we are the only work which specifically tries to answer which in-network signals are the most useful to a generic CCA. There is, however, a significant body of work both in congestion control algorithms which use some form of in-network support [17, 21, 14, 7, 5, 48, 6, 43] and those that specifically use the type of INT signals we target [28, 41]. These approaches use a variety of different signals (link bandwidth, per-hop delay, *etc.*) and reasonably justify these choices. Further, while their goals may be similar, this can result in very different designs (*e.g.,* TIMELY [30] and pFabric [6] both cite minimizing FCT as their goals). However, none of these works systematically address the question of what signals would be best for their (or other) goals.

While these proposed CCAs often build their own infrastructure for collecting in-network signals, [35, 1, 25] propose formats for carrying this data in packet headers. In particular, [35] proposes a general and extensible method for collecting any in-network signals for CCAs.

Meanwhile, the growing interest in these signals builds on a large body of work on in-network telemetry ([49, 18, 29, 27, 26, 47, 32] as a sample). Often this work focuses on collecting metrics for out-of-band analysis and therefore operates on timescales that are not appropriate for congestion control or are not available on a per-path basis. However, the underlying methods for efficiently collecting data at the switch are critical to the per-packet metrics used for CCAs.

There is plenty of existing work in the space of answering general questions about what is possible in congestion control design [46, 9, 8, 11, 38, 23, 4]. These works use very different methodologies (analysis and theorem-proving) than we proposed and accordingly provide quite general insights for their chosen settings/problems. However, most of these works do not aim to answer questions about the quality of the signals available to a CCA. Notably, [4] aims, in part, to determine the signals and other components of a CCA sufficient to achieve a desired property, but is limited to single-flow scenarios and end-to-end signals due to its network model. Further, it primarily focuses on worst-case scenarios rather than average case analysis. We adopt an optimization-based approach to achieve our desired generality properties (independence from the specific control loop) rather than using analysis for both a more descriptive network model and visibility into the full distribution of performance.

We chose Remy [42] as the basis for our CCA generator due to a few desirable properties: (1) there is a significant amount of work available on its capabilities and limitations [42, 44, 39, 3], (2) it produces interpretable CCAs generated in an offline setting, and (3) it can be easily extended/refined. In contrast, some other CCAs which seek to optimize a utility function often perform their exploration online [12, 13], providing no insights across uses for us to analyze. Similarly, many ML-based CCAs may achieve better performance in some cases [20, 44] but employ learning schemes (*e.g.,* DRL) that do not produce an interpretable artifact. Despite this limitation, we experimented with [20] in §A.2 and found it to perform both worse and more inconsistently than R+ in our setting. In addition, some other works combine various methods of learning with existing CCAs to achieve high adaptability to many scenarios [3, 45, 33], but this is not a priority for our methodology (determining the best set of signals for a given set of scenarios).

9 Limitations

The design space our work addressed is infinite, so this paper is not intended to make definitive declarations about in-network signals, but instead is offered as a modest first step in identifying the question of which INT signals are best and proposing a methodology to answer it. Here we outline ways in which our preliminary investigation was limited, and how those limitations could be addressed in future work.

We only trained our CCAs on a few network configurations. Broadening the range of these configurations might lead to new findings, such as perhaps one set of signals generalizes better across a wide range of network conditions than others. In addition, there are many variations of signals that we did not address (*e.g.,* number of flows, different aggregation functions), and perhaps one of these will prove superior to the signals we investigated.

Similarly, we generated CCAs for five different utility functions of the same general form. There exist many other valid forms of utility for which one may want to learn which signals perform best (*e.g.,* message completion time). Likewise, one may be particularly interested in the tail of per-flow performance, rather than the average. While not explored in this work, our framework can accommodate these alternative objectives. Further, we do not evaluate our CCAs in scenarios where they compete with CCAs generated with different utility functions - *i.e.,* we did not have competing utility functions. A model with multiple utilities would more accurately capture the differing goals of flows on real networks, so work in this area could be useful to demonstrate the robustness of the results.

We evaluated our generated CCAs on a few simple topologies and traffic matrices, attempting to extract some important and difficult scenarios for these CCAs. However, the design space here is vast as well. Extensions to evaluate larger topologies (such as a full datacenter network), other difficult scenarios (*e.g.,* incast), and transports with other capabilities (*e.g.,* receiver-driven CCAs [31, 17]) all merit additional investigation if of particular interest to a given network operator.

While we use R+ to produce our lCCAs and achieve independence from any particular control loop, this approach relies on R+'s training algorithm being a good search methodology. We believe it is a reasonable approach and determined that it at least produces significantly better optimization than a naive form of parameter tuning or other learning-based approaches (Appendices A.2). We recognize that there are likely other reasonable approaches to optimizing congestion control algorithms and we encourage exploration of the design space.

References

1 In-band network telemetry (INT) dataplane specification, 2020. https://p4.org/p4-spec/docs/INT_v2_1.pdf.

2 NS-3, 2025. https://www.nsnam.org/.

3 Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. Classic meets modern: a pragmatic learning-based congestion control for the internet. 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, page 632–647, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405892.

4 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)*, pages 951–978, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/agarwal-anup.

5 Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1851182.1851192.

6 Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486001.2486031.

7 Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. Bolt: Sub-RTT congestion control for Ultra-Low latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 219–236, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/arslan.

8 Venkat Arun, Mohammad Alizadeh, and Hari Balakrishnan. Starvation in end-to-end congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 177–192, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544223.

9 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, page 1–16, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3452296.3472912.

10 Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60:58–66, 2017. doi:10.1145/3009824.

11 Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989. URL: https://www.sciencedirect.com/science/article/pii/0169755289900196, doi:10.1016/0169-7552(89)90019-6.

12 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)*, pages 395–408, Oakland, CA, May 2015. USENIX Association. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong.

13 Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-Learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA, April 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/dong.

14 Nandita Dukkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36:59–62, 1 2006. URL: `https://dl.acm.org/doi/10.1145/1111322.1111336`, `doi:10.1145/1111322.1111336`.

15 Gorry Fairhurst and Michael Welzl. The Benefits of Using Explicit Congestion Notification (ECN). Internet-Draft draft-ietf-aqm-ecn-benefits-08, Internet Engineering Task Force, November 2015. Work in Progress. URL: `https://datatracker.ietf.org/doc/draft-ietf-aqm-ecn-benefits/08/`.

16 Sally Floyd. Tcp and explicit congestion notification. *SIGCOMM Comput. Commun. Rev.*, 24(5):8–23, October 1994. `doi:10.1145/205511.205512`.

17 Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 29–42, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3098822.3098825`.

18 Qun Huang, Haifeng Sun, Patrick P. C. Lee, Wei Bai, Feng Zhu, and Yungang Bao. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. 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, page 404–421, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3387514.3405877`.

19 V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA, 1988. Association for Computing Machinery. `doi:10.1145/52324.52356`.

20 Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

21 Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, page 89–102, New York, NY, USA, 2002. Association for Computing Machinery. `doi:10.1145/633025.633035`.

22 Frank P Kelly, Aman K Maulloo, and David Kim Hong Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49:237–252, 1998.

23 Srinivasan Keshav. A control-theoretic approach to flow control. In *Proceedings of the Conference on Communications Architecture &amp; Protocols*, SIGCOMM '91, page 3–15, New York, NY, USA, 1991. Association for Computing Machinery. `doi:10.1145/115992.115995`.

24 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, page 514–528, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3387514.3406591`.

25 Jai Kumar, Surendra Anubolu, John Lemon, Rajeev Manur, Hugh Holbrook, Anoop Ghanwani, Dezhong Cai, Heidi Ou, Yizhou Li, and Xiaojun Wang. Inband Flow Analyzer. Internet-Draft draft-kumar-ippm-ifa-07, Internet Engineering Task Force, September 2023. Work in Progress. URL: `https://datatracker.ietf.org/doc/draft-kumar-ippm-ifa/07/`.

26 Jonatan Langlet, Ran Ben Basat, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. Direct telemetry access. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 832–849, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3603269.3604827`.

27 Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: A better NetFlow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 311–324, Santa Clara, CA, March 2016. USENIX Association. URL: `https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-yuliang`.

28 Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3341302.3342085`.

29 Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 101–114, New York, NY, USA, 2016. Association for Computing Machinery. `doi:10.1145/2934872.2934906`.

30 Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *SIGCOMM Comput. Commun. Rev.*, 45(4):537–550, aug 2015. `doi:10.1145/2829988.2787510`.

31 Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: a receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 221–235, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3230543.3230564`.

32 Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. SketchLib: Enabling efficient sketch-based monitoring on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 743–759, Renton, WA, April 2022. USENIX Association. URL: `https://www.usenix.org/conference/nsdi22/presentation/namkung`.

33 Lorenzo Pappone, Alessio Sacco, and Flavio Esposito. Mutant: Learning congestion control from existing protocols via online reinforcement learning. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 1507–1522, Philadelphia, PA, April 2025. USENIX Association. URL: `https://www.usenix.org/conference/nsdi25/presentation/pappone`.

34 K. K. Ramakrishnan and Raj Jain. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 303–313, New York, NY, USA, 1988. Association for Computing Machinery. `doi:10.1145/52324.52355`.

35 Abhiram Ravi, Nandita Dukkipati, Naoshad Mehta, and Jai Kumar. Congestion Signaling (CSIG). Internet-Draft draft-ravi-ippm-csig-00, Internet Engineering Task Force, August 2023. Work in Progress. URL: `https://datatracker.ietf.org/doc/draft-ravi-ippm-csig/00/`.

36 Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. CUBIC for Fast Long-Distance Networks. RFC 8312, February 2018. URL: `https://www.rfc-editor.org/info/rfc8312`, `doi:10.17487/RFC8312`.

37 Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. 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, page 735–749, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3387514.3405899`.

38 S. Shenker. A theoretical analysis of feedback flow control. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols*, SIGCOMM '90, page 156–165, New York, NY, USA, 1990. Association for Computing Machinery. `doi:10.1145/99508.99547`.

39 Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 479–490, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2619239.2626324`.

40 W. Richard Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, January 1997. URL: `https://www.rfc-editor.org/info/rfc2001`, `doi:10.17487/RFC2001`.

41 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)*, pages 255–274, Boston, MA, April 2023. USENIX Association. URL: `https://www.usenix.org/conference/nsdi23/presentation/wang-weitao`.

42 Keith Winstein and Hari Balakrishnan. Tcp ex machina: computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 123–134, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2486001.2486020`.

43 Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, and Shivkumar Kalyanaraman. One more bit is enough. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, page 37–48, New York, NY, USA, 2005. Association for Computing Machinery. `doi:10.1145/1080091.1080098`.

44 Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, Boston, MA, July 2018. USENIX Association. URL: `https://www.usenix.org/conference/atc18/presentation/yan-francis`.

45 Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 255–274, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3603269.3604838`.

46 Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. Axiomatizing congestion control. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(2), June 2019. `doi:10.1145/3341617.3326148`.

47 Yang Zhou, Ying Zhang, Minlan Yu, Guangyu Wang, Dexter Cao, Eric Sung, and Starsky Wong. Evolvable network telemetry at facebook. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 961–975, Renton, WA, April 2022. USENIX Association. URL: `https://www.usenix.org/conference/nsdi22/presentation/zhou`.

48 Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2785956.2787484`.

49 Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 479–491, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2785956.2787483`.

A Appendix

A.1 Modification Details

As mentioned in §3, changes to the component systems in our methodology were necessary.

A.1.1 Remy Modifications (R+)

We modify Remy primarily by (1) increasing the network configuration sample size, (2) adding in-network signal support, and (3) modifying the action types. The default sample size in [42] is 16 networks, but we found that > 50 were needed for consistent results (we used 200 in our evaluations). Adding in-network signals (2) was primarily achieved by adding additional fields to the packets. In the Remy model, there is a single bottleneck, so the queue length and the recent link utilization are simply added as the packet is put on the wire. Lastly, we found that we needed to modify the action types to get better/consistent performance (3).

Specifically, in its default form, RemyCC rules map signal space to actions in the form of a cwnd multiplier, a value to add to the cwnd, and the pacing rate. We found that setting the pacing rate directly was problematic. Rules that were less commonly-used and covered a large space of signals were not served well with a single pacing rate. As an example, a rule that covers a signals space in which the sending rate and the receiving rate ranges are roughly equal (indicating no congestion), but large (covering receiving rates in a wide range), one cannot set a sending rate that will be optimal for all scenarios covered by this rule. Senders with sending/receiving rates near the upper bound of the range of the rule will be better served with a faster pacing rate than those at the other extreme. In response to this, we modified Remy so that the pacing rate is a function of the current delivery rate. This resulted in a large increase in performance for network configs which used such rules, and therefore cause an overall increase.

The performance difference between the results from default Remy and R+ are shown in Figure 16 for default utility and E2E signals. The gains relative to default Remy for (1) adding just the additional training scenarios and (2) also changing the rate to be a function of delivery rate are shown (R+). As shown in the figure, adding more training scenarios dramatically increased (up to 56%) the performance in many scenarios, while decreasing it in others. This is expected as the baseline form of Remy would overfit to the scenarios it *did* train on, doing particularly well in those at the cost of others. While not in the figure, we found that this change was critical to seeing consistent performance across generations. With the generalization of the sending rate as a function of the receiving rate ("Rate Function" in the figure), *no* scenarios result in worse performance than baseline Remy and over 10% improve by over 50%. Again, this difference can be attributed to the generated CCA not having to choose some subset of scenarios to excel in, and it can instead optimize over the entire training space.

A.1.2 ns-3 Modifications

In order to get RemyCCs to work in ns-3, we had to make a few modifications to the default TCP socket in ns-3. In particular, our three major modifications were to (1) turn on the timestamp option and increase its resolution from ms to μs, (2) change how the pacing rate is updated, and (3) add hooks to clear state. Since Remy trains its CCAs on a very simple network model (with very high-specificity timestamps), the RTT measurements used by default in a TCP socket were too inaccurate and smoothed too much to be useful to the

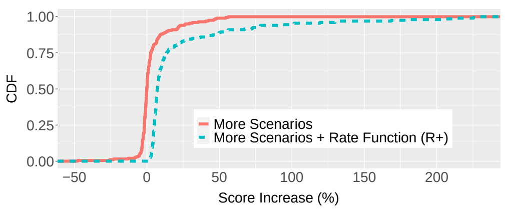


Figure 16 Distribution of performance improvement (% score increase) from Remy to R+ for the E2E signal set under default utility.

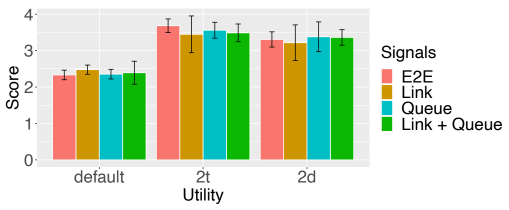


Figure 17 Average scores across utilities and signal sets for PCC-RL [20].

RemyCCs. Accordingly, we implemented high-specificity timestamps (1) and exclusively used these for the RTT measurements used by the RemyCC. We believe such timestamps are not impractical as recent work has implemented systems which gather precise delay measurements [24]. Similarly, another discrepancy exists in the pacing rates. Remy implements its pacing by ensuring that when a packet is ready to be sent that enough time has passed since the last send according to the current pacing rate. ns-3 sockets, however, use timers which are set to wait $\frac{1}{rate}$ time to send again when a packet is sent. Accordingly, changes to the pacing rate made between packet sends will not have impact until the next packet is sent (causing the flow to wait much longer than the RemyCC intended based on its training). To resolve this, we simply recalculate the time left on the pacing timer when the pacing rate is changed (2). Lastly, Remy trains its CCAs on exponentially distributed flow lengths where each sender samples for on and off times. This is similar to the On/Off Application in ns-3, but does not retain socket state. Accordingly, we added hooks to the TCP socket to notify it that it should clear the Remy state (3).

A.2 PCC-RL Results

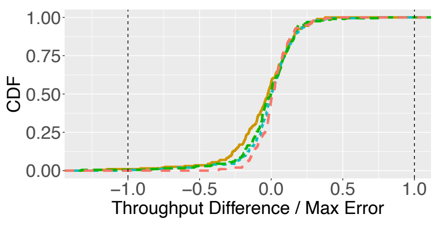
In an attempt to find the best CCA generator for our framework, we also ran experiments with PCC-RL [20]. All other parameters in our framework remained the same: the signal sets, utility functions, and network configurations. The only change was the optimizer that produces the lCCAs. Using [20], we trained models that use our signal sets and result in changes to the CWND and pacing rate. We then evaluated these models in ns-3 in the same scenarios that we evaluated the lCCAs from R+.

The results for default, 2d, and 2t utility are shown in Figure 17. Notably, the two primary results from R+ remain the same: all signal sets perform within average error of one another. INT signals did not clearly help overall utility and no single signal clearly outperforms the other. However, in contrast to R+, the error is overall much larger and the scores are overall lower (*e.g.,* R+ achieves scores around 3.4 for default utility).

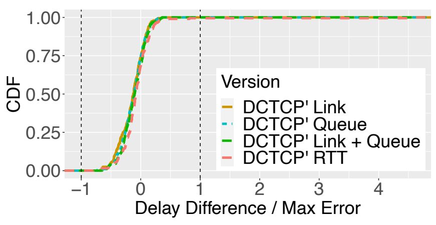
In light of these results, we chose to continue with R+ as our optimizer. We believe these large differences to be due to (1) our modifications to Remy and (2) PCC-RL training with only one flow (thus making optimizing overall utility more difficult). Perhaps multi-agent RL approaches could improve PCC-RL's relative performance, but we leave such investigations to future work.

A.3 DCTCP Modifications

Similarly to §7, we attempted to apply our findings to DCTCP [5]. DCTCP uses ECN bits in order to create an EWMA that estimates the current degree of congestion, α. The

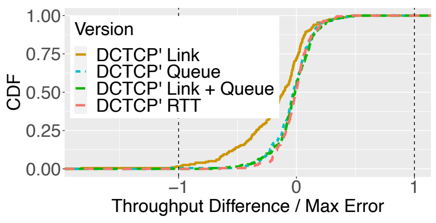


(a) Throughput

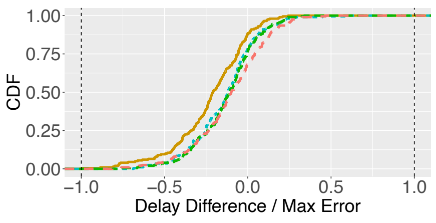


(b) Delay

Figure 18 Distribution of significant throughput and delay differences between DCTCP' versions and baseline DCTCP.



(a) Throughput



(b) Delay

Figure 19 Distribution of significant throughput and delay differences between DCTCP' versions and baseline DCTCP on a line topology.

window size is then adjusted based on the value of α. In baseline DCTCP, received bytes are generally recorded as "experienced congestion" if the ECN bit is set. For this baseline, we use the guidance for picking the ECN marking threshold in the switches from [5]. In our experiments with other congestion signals, we only modify the definition of bytes that experienced congestion. We refer to all versions using different signals than ECN for this calculation as DCTCP'. Rather than using ECN, we try classifying packets based on thresholds of INT signals and end-to-end delay. The results are shown in Figure 18. In the figure, we see that we can achieve similar performance to the baseline algorithm (which uses a simple form of INT, ECN) with just RTT measurements. Similarly, marking bytes as congested based on INT does not provide any significant performance benefit. This is perhaps unsurprising as setting the queue-based threshold to the ECN marking threshold should be effectively equivalent behavior. Link signals, however, required more careful tuning to achieve the same performance of the baseline.

Once again, these (manually tuned) results do not attempt to conclusively determine that INT signals could not have helped the performance of DCTCP, However, these results do definitively show that in this regime, *both* INT and RTT can produce effectively the same performance as ECN.

We ran the same experiments on our line topology with extra senders enabled to create multiple bottlenecks. The measured throughput and delay are shown in Figure 19. As with the simpler dumbbell topology, all DCTCP' are able to achieve the same performance as the baseline version.