Automating network heuristic design and analysis

Anup Agarwal†, Venkat Arun§, Devdeep Ray†, Ruben Martins†, Srinivasan Seshan†
†Carnegie Mellon University, §MIT CSAIL

ABSTRACT

Heuristics are ubiquitous in computer systems. Examples include congestion control, adaptive bit rate streaming, scheduling, load balancing, and caching. In some domains, theoretical proofs have provided clarity on the conditions where a heuristic is guaranteed to work well. This has not been possible in all domains because proving such guarantees can involve combinatorial reasoning making it hard, cumbersome and error-prone. In this paper we argue that computers should help humans with the combinatorial part of reasoning. We model reasoning questions as $\exists\forall$ formulas [1] and solve them using the counterexample guided inductive synthesis (CEGIS) framework. As preliminary evidence, we prototype CCmatic, a tool that semi-automatically synthesizes congestion control algorithms that are provably robust. It rediscovered a recent congestion control algorithm that provably achieves high utilization and bounded delay under a challenging network model. It also found previously unknown variants of the algorithm that achieve different throughput-delay trade-offs.

CCS CONCEPTS

• **Networks** → **Protocol correctness**; *Transport protocols*;
• **Theory of computation** → **Automated reasoning**;

KEYWORDS

Automated reasoning, Congestion control

ACM Reference Format:
Anup Agarwal†, Venkat Arun§, Devdeep Ray†, Ruben Martins†, Srinivasan Seshan†. 2022. Automating network heuristic design and analysis. In *The 21st ACM Workshop on Hot Topics in Networks (HotNets '22), November 14–15, 2022, Austin, TX, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3563766.3564085

1 INTRODUCTION

Heuristics permeate computer systems, including congestion control, traffic engineering, CPU/cluster scheduling, adaptive bit rate (ABR) algorithms, load-balancing, and caching. For some heuristics and domains, we have formal guarantees on what heuristics are best for certain workloads, or environment assumptions (e.g., work-stealing for scheduling [11] and, LRU for cache replacement policies [2]). These theoretical results are backed up by the popularity of these algorithms in practice [46]. However, for most heuristics and domains, we either do not have such guarantees, or the guarantees are proven under unrealistic assumptions. In some areas, lack of clarity has led to hundreds of papers with promises of improved performance. In this paper, we ask "What would it take to obtain formal guarantees in these areas?".

Two challenges make this hard. First, heuristics operate in many environments, and balance multiple objectives. It is laborious to analyze and obtain guarantees for a large number of environment/objective pairs. For instance, in congestion control[1], the environments include various network types (e.g. wired [15, 35], cellular [67, 72], satellite [13], datacenter [3, 73], networks with explicit feedback [31, 39]), and CCAs balance between an often conflicting set of objectives, e.g., throughput [35], delay [12, 15], co-existence/fairness [7, 27, 32], priority [48, 58], and flow/co-flow completion time [4, 18, 23], to cater to the diverse needs of different applications.

Second, the environment and heuristics can interact in complex ways. Reasoning about them can be cumbersome and error-prone. For instance, in congestion control, one needs to carefully consider subtleties of duplicate acknowledgements (ACKs), timeouts, ACK aggregation, delayed ACKs, token bucket filters, and transmission timing jitter that are common in real networks [6].

Both challenges above pertain to the combinatorial explosion of possibilities when reasoning about systems. This is where computers shine [28] We envision a human-computer collaboration where computers do combinatorial reasoning to broadly answer two types of questions, (a) given the environment assumptions, design an algorithm that provably achieves some given desired properties under the environment, and (b) given an algorithm, generate assumptions about the environment under which the algorithm is guaranteed to achieve given desired properties. Humans can iteratively update the desired properties, and/or the search space for algorithms and assumptions depending on their use-case.

In this paper, we explore the feasibility of our human-computer collaboration approach by designing a tool, CCmatic, that automates reasoning about congestion control

[1]We use congestion control as the running example in this paper. §5 discusses other domains.

HotNets'22, November 14–15, 2022, Austin, TX, USA

ACM ISBN 978-1-4503-9899-2/22/11.
https://doi.org/10.1145/3563766.3564085

algorithms. We model reasoning questions as ∃∀ logical formulas, and use the counter-example guided inductive synthesis (CEGIS) framework to solve the formulas [1, 61]. The CEGIS framework involves iterative interactions between a generator and a verifier. The generator proposes a candidate solution from a defined search space, while the verifier produces a counterexample that breaks the proposed candidate.

We overcome several technical challenges to tractably solve our formulas (§3). For instance, we carefully construct the generator's search space to ensure it captures a variety of possible solutions while keeping it simple enough to keep the search tractable. Further, CEGIS is prone to enumerating the search space as each counterexample eliminates only a few candidate solutions [1, 59]. We use domain specific insights to encode more information about why a counterexample breaks candidate solutions, allowing us to prune a larger part of the search space per counterexample. Our optimizations collectively improve solving time by at least 60×. While domain specific, our optimizations have mathematical equivalents and can be applied to other domains as well (§3.1.2, §5).

We show that automation can help quickly explore large design spaces. We ask CCmatic to find CCAs that provably achieve high utilization and bounded delay under a recently proposed network model [6]. CCmatic was not only able to generate algorithms that matched existing non-intuitive designs [24, 63], it was able to produce several variants of this design. These variants provide a new range of throughput-delay trade-offs that the existing design did not explore.

Outline. We elaborate the reasoning queries that we hope to answer (§2), describe our approach, technical challenges, and salient features of our design (§3), and our results (§4). We believe that our approach can answer other reasoning queries within congestion control, and also answer similar queries in other domains like adaptive bit-rate streaming (ABR), and scheduling. We discuss anticipated challenges and ideas to generalize our work (§4.1, §5).

2 VISION AND SCOPE

Designing and analyzing heuristics is a difficult task. Many prior works have tried to address this problem. For the design part, prior work used techniques such as optimization [66], analytical modelling [45, 65], and reinforcement learning [44, 69]. For analysis part, techniques such as fuzzing/testing [14, 37, 62], and benchmarking frameworks [52, 70], have been proposed. A recent work, CCAC [6], proposed the use of formal techniques to verify whether a given congestion control algorithm (CCA) satisfies a given desirable property. Formal techniques have the advantage of providing provable guarantees. Inspired by CCAC, we investigate the use of formal techniques to automate important steps in the methodology of design and analysis of heuristics.

We believe the following steps can be (partially) automated: (1) synthesizing heuristics, (2) identifying assumptions, (3) differential comparison. Our approach is to frame and solve automation questions as ∃∀ formulas.

Synthesizing heuristics. We seek to synthesize heuristics that provably ensure certain desired properties under a specified environment. Synthesizing a heuristic means deciding what controllable actions should be taken in response to observable signals. For instance for a CCA, we synthesize congestion window/rate changes in response to measurable signals like acknowledgements, delays, and losses.

This was one of the motivations of CCAC, where developers can iteratively query CCAC to obtain counterexamples that can inform their CCA design. Designing robust CCAs that work under all target circumstances is non-trivial, so we reduce the developer's effort by formulating and automatically answering the *CCA synthesis query*, "does there exist a CCA such that for all realistic network traces (e.g., those allowed in the CCAC model), the CCA achieves the given desired properties (e.g. high utilization and/or bounded delay)". Program synthesis techniques have recently been used to reverse engineer CCAs from network traces [26]. Encouraged by them, we apply similar techniques to synthesize (possibly novel) CCAs that provably ensure desired properties.

Identifying assumptions. Designs and implementations of heuristics often make implicit assumptions about the environment they operate in. For instance, Copa [7] assumes that queuing delays are close to zero under low utilization [6].

Uncovering such implicit assumptions is hard. Existing techniques like fuzzing/testing and even CCAC, produce concrete counterexamples where heuristics fail. These counterexamples are often hard to interpret. Instead, we seek to synthesize assumptions as logical constraints that (1) serve as a high level description of equivalence classes of counterexamples and (2) are human interpretable.

Formally, we ask "does there exist an assumption such that for all system traces, the system trace ensures given desirable properties iff the trace satisfies the assumption". Synthesized assumptions take the form of logical constraints on system's environment, e.g., "a network can delay packets by at most $100\mu s$". Such logical constraints are easier to interpret than an execution trace of a system.

Differential comparison. We can formally perform differential comparison between heuristics by asking queries like: "given CCA *A*, CCA *B*, and some desirable properties, for all networks on which CCA *A* ensures the desirable properties, what are additional network constraints are needed for CCA *B* to ensure the properties". Such queries are useful for system operators to decide what heuristic they might want to deploy in their custom system. Again, similar to queries on identifying assumptions, differential comparison queries will give us logical constraints that are human interpretable and capture a set of network traces as opposed to individual network traces that a tool like CCAC might generate.

In this paper we provide preliminary results for solving the CCA synthesis query. We believe our approach can be

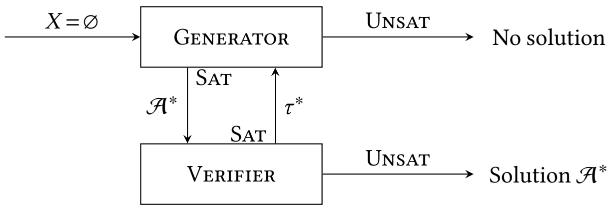

Figure 1: CEGIS loop (figure adapted from [1]).

generalized for identifying assumptions and performing differential comparison (§4.1). We also see this as an important step towards automated reasoning for other domains (§5).

3 DESIGN SKETCH

We model the queries we want to answer as ∃∀ logical formulas. Formulas with quantifiers are typically harder to solve than quantifier-free formulas (∃∀ formulas are in Σ_p^2 complexity class, while quantifier-free formulas are in NP). There are multiple techniques in the formal methods literature to solve such formulas [20, 30, 55–57] many of which are implemented by solvers for quantified formulas (e.g., Z3 [19], CVC5 [9]), but they are often slow out-of-the-box [10]. We explore the use of counterexample guided inductive synthesis (CEGIS) [1, 61] framework as it allows incorporating domain specific insights to speed up formula solving.

Primer on CEGIS. CEGIS is an iterative approach involving interactions between a generator and a verifier. It takes as input a formula of the form $\exists\mathcal{A}.\forall\tau.\sigma(\mathcal{A},\tau)$, where $\sigma(\mathcal{A},\tau)$ is a quantifier free formula. Then, the generator tries to synthesize a candidate solution $\mathcal{A}^*$, and the verifier tries to find a counterexample τ^* that breaks the solution. In our running example of CCA synthesis, $\mathcal{A}$ is the CCA we hope to find, τ is a network trace, and the specification σ encodes whether the trace τ is realistic under the CCA $\mathcal{A}$ and whether τ satisfies the desirable properties. The generator finds $\mathcal{A}$ in a search space typically defined using a template with parameters or holes. The generator assigns values to the parameters to obtain a concrete CCA. For instance, a filled template may specify how the CCA's congestion window (cwnd) is updated based on historical cwnd, packet acknowledgements (ACKs), and events such as losses or timeouts.

CEGIS Workflow (Figure 1). The generator proposes a candidate CCA, $\mathcal{A}^*$. Initially, this is an arbitrary choice. The verifier checks whether our specification can be violated by checking if $\neg\sigma(\mathcal{A}^*,\tau)$ is satisfiable. If so, the verifier produces a counterexample trace τ^* that violates the specification. We add τ^* to the set of counterexamples, X. Now, the generator searches for a CCA that can ensure the specification under all counterexamples till now, i.e., it finds $\mathcal{A}$ such that $\forall\tau^*\in X.\sigma(\mathcal{A},\tau^*)$ is true. Note, the generator only checks over the finite (and hopefully, small) set X. This loop ends in two cases, (1) the verifier fails to find a counterexample, proving that under the latest $\mathcal{A}^*$, the specification can't be violated, thus $\mathcal{A}^*$ is a solution to the ∃∀ query, or (2) the generator fails to find a CCA, this means that there is no solution to the query in the search space for $\mathcal{A}$. Hence this method is both sound and complete.

The verifier and generator can be implemented using different techniques including machine-learning [38], and constraint solving [1, 26], or a mix of both [25]. We use constraint solvers as their solutions are more interpretable and we can logically encode our requirements.

Challenges. There are several challenges in applying CEGIS that can be categorized into (1) encoding and (2) scalability.

Encoding. It is non-trivial to precisely specify the CCA template and desired properties. Ideally, we want a template that is expressive enough to capture a variety of actions that CCAs can take, at the same time, we want to restrict the template to keep our search tractable. Likewise, we want our desired properties to be strong enough to synthesize potentially novel CCAs, but keep them relaxed enough so that solutions to our queries exist.

Scalability. The CEGIS approach can be slow in general. The generator/verifier formulation can often involve non-linearities consuming significant time per iteration. CEGIS loop is also prone to enumerating the generator's entire search space through many iterations of the loop since each counterexample eliminates few candidate solutions [1, 59].

We describe how we use CEGIS to solve the CCA synthesis query, and our learnings to overcome above challenges.

3.1 Prototype: CCmatic

Recall, we model the CCA synthesis query as $\exists\mathcal{A}.\forall\tau.\sigma(\mathcal{A},\tau)$. The CCA, $\mathcal{A}$, controls when packets are sent in response to previous network behavior. The trace τ specifies when packets are dropped or delayed. We model σ as:

$$\sigma(\mathcal{A},\tau) := feasible(\mathcal{A},\tau) \implies desired(\mathcal{A},\tau) \qquad (i)$$

i.e., all feasible traces must satisfy our desired properties (e.g., "high utilization AND low delay"). Here, feasiblity means two things, (1) packets in the trace should be sent according to the CCA, (2) packet should be ACKed or dropped as allowed by a realistic network. To encode what traces are $feasible$ and whether they satisfy our *desired* properties, we use the encoding proposed by CCAC [6]. CCAC models the network environment using Network Calculus [41], and captures a wide variety of sub-RTT phenomena that real networks exhibit.

We encode the generator as a constraint satisfaction problem in the theory of quantifier-free linear real arithmetic (QF-LRA) [40], and use Z3 [19] to solve it. We directly use CCAC [6] as the verifier.

3.1.1 *Encoding challenges.*

Template for CCA. CCAs are defined by how they handle different network events such as ACKs, delays, and losses. As an initial attempt, we consider lossless networks with infinite buffers. The CCA template sets the cwnd every round trip time (RTT) in response to ACKs. Prior work has shown CCAs operating on summary metrics every RTT to be as good as fine-grained, per-ACK control [6, 49]. This fits well with CCAC's abstraction of available control actions and response granularity, allowing us to easily encode the templated candidate solution into CCAC's formulation.

CCAs can also define their own state about historical events. Allowing such state in the template significantly increases the size and complexity (non-linearity) of the encoding constraints and slows down synthesis time. Instead, we give direct access to a small period of historical information to the CCA. In summary, our template is:

$$cwnd(t) = \sum_{i=1}^{h} \Big(\alpha_i cwnd(t-i) + \beta_i ack(t-i) \Big) + \gamma \qquad (ii)$$

where, α_i, β_i, and γ are coefficients (holes or parameters) synthesized by the generator, $cwnd(t)$ is the cwnd at time t, $ack(t)$ is the cumulative bytes ACKed by time t, and h is the number of historical RTTs that the CCA can query statistics about. The time indices (t), $(t-i)$ are in units of propagation delay. The user can experiment with different templates (see §4.1) to explore the design space.

Steady state behavior and desired properties. Constraint solvers work best with finite traces and it is hard to directly reason about steady state behavior. CCAC tackles this by letting the constraint solver pick arbitrary initial values for cwnd and queue size, effectively treating the time before $t = 0$ to have had arbitrary evolution of the packet delays, drops, and other network behavior.

This creates an issue—a desired property (e.g. "high utilization AND bounded delay") can be violated in a finite trace with arbitrary initial conditions. For instance, at the start of a flow, the CCA needs to ramp up before it can achieve high utilization. Likewise, the trace may start with a large initial queue that can cause excessive delays. In such situations, the best any CCA can do is to increase or decrease the cwnd in the right direction. Thus, we relax the original desired property as: "(the CCA should have a high utilization OR increase its cwnd) AND (it should maintain a small queue OR reduce its cwnd)". If this property is true, mathematical induction proves that in a long-running trace, the synthesized CCA either achieves our original desired properties directly, or moves in a direction to realize our original desired properties. Specifically, the encoding we use is $ack(T) - ack(0) \geq \text{thresh}_U * C * T$ (high utilization), $cwnd(T) > cwnd(0)$ (increase cwnd), $cwnd(T) < cwnd(0)$ (decrease cwnd), $\forall t. queue(t) \leq \text{thresh}_D$, where T is the duration of CCAC's trace and C is the link rate. We can vary the desired utilization (thresh_U) and delay thresholds (thresh_D) (see §4).

Note, a finite trace in CCAC has fixed *average* link rate, we incorporate variable link rates using CCAC's jitter term and mathematical induction (as was done in CCAC [6]).

Putting it all together. To summarize, we first define a template of what the CCAs look like. The generator picks a CCA from this template. The verifier either certifies that this CCA satisfies the desired properties or produces a concrete network trace that "breaks" the CCA. The generator produces another CCA that is not broken by any of the verifier-produced traces thus far until either a solution is found or the generator proves that none of the CCAs specified by its template can work.

3.1.2 Scalability challenges. The speed of the CEGIS loop is affected by (1) time per iteration, and (2) number of iterations. We reduce both these factors to improve solving time.

Time per iteration. The generator formulation has non-linear constraints. These mainly involve the product between two generator variables, one of which is a coefficient variable. Specifically, the *cwnd* function (Equation *ii*) involves product between old *cwnd* and a coefficient. The old *cwnd* in turn depends on coefficients. We restrict coefficients to take values from a discrete set. This allows us to convert the product terms into linear terms using "if then else" constraints. We replace the product term $v * u$ as $\sum_{a \in A} ite(v == a, a * u, 0)$ where A is the set of possible values for v and $ite(c, texpr, fexpr)$ evaluates to $texpr$ if condition c is true and $fexpr$ otherwise.

Number of iterations. Each counterexample might only eliminate few candidate solutions in CEGIS. We reduce the iterations by (1) encoding more information about why a particular candidate CCA did not work, allowing us to prune a range of candidate CCAs (range pruning), and (2) producing network traces that are likely to break the most number of candidate CCAs (worst-case counterexample).

Problem. The generator tries to find a CCA, $\mathcal{A}$, such that $\forall \tau^* \in X. \sigma(\mathcal{A}, \tau^*)$. Recall, $\sigma(\mathcal{A}, \tau^*)$ is $feasible(\mathcal{A}, \tau^*) \implies desired(\mathcal{A}, \tau^*)$. To satisfy σ the generator can, (1) make $desired(\mathcal{A}, \tau^*)$ true, or (2) make $feasible(\mathcal{A}, \tau^*)$ false. The latter is easy. The generator can simply tweak the CCA so it has a different behavior than any trace in X. This forces the verifier to produce a new trace for each slight variation of the CCA which is inefficient (§4 shows the number of iterations for various methods). This is a common problem with CEGIS [59, 60].

Range Pruning. The problem is that each trace in X eliminates exactly one CCA behavior. Our solution is for each trace to eliminate a range of behaviors, which makes $feasible$ harder to falsify with trivial tweaks. Thus the generator will spend more iterations satisfying $desired$.

For completeness we mention how we map an exact trace produced by CCAC to a range of possible CCA behaviors. We omit detailed derivation due to space limitations. According to notation used in the CCAC paper [6], the range of CCAs for a trace is such that cumulative bytes sent by the CCA (A_t) lies in the interval $[S_t, \infty]$ if $W_t = W_{t-1}$ and interval $[S_t, Ct - W_t]$ otherwise. These bounds can be derived by simple algebraic manipulation of the constraints in the CCAC paper. Here S_t and W_t are produced by the verifier. If the corresponding A_t produced by the generated CCA lies within this range, $feasible$ is true.

Worst-case counterexample. We can further improve the range of CCAs pruned. The range is specified using an upper and lower bound, e.g., $[S_t, Ct - W_t]$. If the upper and lower bounds are close to each other, then few CCAs are captured by the trace. To maximise the range of CCAs captured, we ask the verifier to find a trace that maximizes the minimum range for any timestep, i.e., find a trace that maximizes $min_t(u_t - l_t)$ where $[l_t, u_t]$ is the range for cwnd at time t in the trace. We maximize using binary search. This involves calling the verifier multiple times in a single CEGIS iteration. For us, verifier calls are typically fast. Hence, the reduction in iterations makes up for extra time spent in verifier calls (§4).

For intuition, consider that most candidate CCAs will not work even on ideal links. The verifier has to do very little to break them. By asking it to maximize the range of CCAs eliminated, we can quickly get to the interesting candidate CCAs that are harder to break.

Our optimizations do not violate soundness or completeness of the logic, i.e., the solution set does not change on adding the range pruning and worst-case counterexample optimizations. We merely avoid otherwise redundant iterations. Further, while we described our optimizations in the context of CCA synthesis, these optimizations have mathematical equivalents and can be applied in a domain agnostic manner. For instance, range pruning is similar to variable movement [54] or adding existential quantifiers for dependent inputs [34, 59].

4 RESULTS

We study the solutions produced CCmatic, and solving time.

Methodology. We ask CCmatic to synthesize CCAs that achieve high utilization and bounded delay in steady state in a lossless network. We consider two different search spaces, (1) without access to historical *cwnd* (i.e., coefficient of *cwnd*, β_i, is fixed to 0 for all i), and (2) with access to historical *cwnd*. We consider two domains for the coefficients and constants: (1) small: $\{-1,0,1\}$, and (2) large: $\{-2,-3/2,-1...,2\}$ or $\{\frac{i}{2} : |i| \leq 4 \land i \in \mathbb{Z}\}$. The small domain restricts to additive responses, while the large domain includes multiplicative responses. We explore solutions that use up to 3 RTTs of historical information (by setting $h = 3+1 = 4$). We let CCAC jitter each packet up to 1×RTT. We set requirements as "≥ 50% utilization AND ≤ 4×RTT delay", and later vary these thresholds. CCAC found traces where BBR [15], Copa [7] achieve arbitrarily low utilization, so we start with 50% utilization as a reasonable goal [6].

Synthesized CCAs. One of the solutions we find is: $cwnd(t) = ack(t-1) - ack(t-3) + 1$. This CCA, called RoCC, was recently proposed [24, 63]. On each RTT, it sets cwnd as bytes acknowledged in last 2 RTTs plus a small additive increment. On an ideal link with constant rate, RoCC converges to a queue of BDP + MSS (bandwidth-delay product + maximum segment size) bytes. In the CCAC model, for the same choice of parameters we use in this paper (i.e., jitter = 1×RTT), if this additional queuing is not present we risk getting arbitrarily low utilization [5]. An explanation for why this simple rule works is available [63].

Extensions. We ask CCmatic to produce all possible solutions, implying that there are no other solutions in our search space apart from those produced by CCmatic. In the search space without historical cwnd, in the large domain space, we find a total of 12 CCAs that meet our requirements. This is an exhaustive set out of 9^5 candidate solutions in the search space. 9 possible values for each decision variable, and 5 variables (4 coefficients and 1 constant). The 12 synthesized CCAs use different amount of historical knowledge. Six of them use information about last 2 RTTs, and other six use last 3 RTTs. All these 12 CCAs are minor variations of RoCC, e.g.,

Params	Domain	Search size	Baseline # Itr	Baseline Time	RP # Itr	RP Time	RP+WCE # Itr	RP+WCE Time
No cwnd	Small	3^5	100	3m	30	30s	7	3s
No cwnd	Large	9^5	DNF	DNF	60	1m	50	1m
cwnd	Small	3^9	DNF	DNF	100	9m	50	30s
cwnd	Large	9^9	DNF	DNF	360	32h	80	45m

Table 1: Time to synthesize first solution. DNF: did not finish within a week, # Itr: number of iterations, RP: range pruning, WCE: worst-case counterexample, (h, m, s): (hours, minutes, seconds). # Itr and time are rounded.

$$cwnd(t) = \frac{3}{2}ack(t-1) - \frac{1}{2}ack(t-2) - ack(t-3) \qquad (iii)$$

An interesting observation is how the solution space changes as we change the utilization and delay thresholds. At ≤ 4×RTT delay, if we require CCAs to have ≥ 65% utilization, only 2 CCAs remain. With ≥ 70% utilization, only 1 CCA remains (Equation *iii*). At ≥ 50% utilization, we get 245 solutions with ≤ 8×RTT delay, 9 solutions with ≤ 3.6×RTT delay and no possible solution with ≤ 3×RTT delay.

Scalability. Table 1 shows improvement in synthesis time from various optimizations. All runs use the encoding techniques described in §3.1.1, they only differ in the optimizations described in §3.1.2. We terminate the CEGIS loop after finding the first solution. All runs were done on server machine with Intel Xeon Gold 6226R CPU (32 physical cores) and 256 GB RAM using Z3 version 4.8.17.0. CCmatic uses only 1 core at a time.

Our optimizations improve synthesis time by at least 60×. The optimizations are essential for applying our approach as the loop does not converge even after a week of running in many cases with the baseline. Further, the baseline is even slower than a brute-force search (due to generator overheads). The complexity of verifier formulation is fixed across iterations, unlike the generator that gets more constraints in each iteration. The verifier typically takes ≈ 0.5s to compute a counterexample. A brute force search where the verifier is called for each candidate solution over a search space with size 3^5 would take ≈ 120s, while the baseline takes ≈ 180*s* (3m in Table 1). However, such brute force would take more than 6 core × years of computing time for a search space of size 9^9, whereas our approach can find a solution in 45m *using a single core.*

4.1 Next steps

We considered CCA synthesis with lossless networks, and utilization/delay objectives. We discuss next steps in expanding to other environments, objectives, and queries.

Environment and objectives. For lossless networks, a simple CCA template sufficed. This template may not suffice for lossy networks and/or fairness/co-existence objectives. A natural fix is to encode *cwnd* functions with conditionals, i.e., **`if`** *cond* **`then`** $cwnd \leftarrow expr_1$ **`else`** $cwnd \leftarrow expr_2$, where *cond*, $expr_1$, and $expr_2$ are decided by the generator (similar to Equation *ii*). This template expresses traditional CCAs, e.g., for AIMD [16], *cond* is loss detected, $expr_1$ is multiplicative decrease, and $expr_2$ is additive increments. This template substantially increases our search space size. We envision synthesizing subsets of the expressions at a time instead of all at once, and/or have coefficients for known good signals instead of having coefficients for each observable quantity.

We hope to use CCmatic to solve open problems. Recent work [5] showed that network delays can cause competing flows to starve for many known CCAs including BBR [15], Cubic [35], and PCC [22, 48]. It is unknown if a CCA outside this class can avoid starvation.

Other queries. In §2, we discuss identifying assumptions and differential comparison. Both find an assumption and require describing a template of an assumption. A simple template could just be a set of parameterized inequalities (similar to [40]). However, it is challenging to define the specification σ. Say we require an assumption such that "a trace satisfies desired properties *if and only if* the trace satisfies the assumption" (from §2). It might be too harsh to synthesize an assumption that is both necessary and sufficient. Such an assumption may not even exist in general, let alone in our search space. Ideally, we want the weakest sufficient assumption. Simply querying for a sufficient assumption causes the CEGIS loop to trivially output "False", since the assumption "False" satisfies the sufficiency requirement (i.e., the *if* part). To solve this, we are exploring three approaches: (1) techniques like MaxSAT [8] to define the weakest sufficient assumption, (2) re-defining our template as feasible actions that the network can take instead of constraints that the trace satisfies, (3) weakening the necessary and sufficient requirement, e.g., "if trace satisfies assumption then utilization $\geq 70\%$ else utilization $\leq 50\%$."

5 GENERALIZING TO OTHER DOMAINS

We describe what would it take to apply our approach to a new domain, then for each domain we describe why automated reasoning is a good fit, what (open) questions in those domains could fit within our framework, and any unique domain-specific challenges we anticipate.

The CEGIS approach requires a verifier. For congestion control, we were able to use prior work (CCAC [6]). Building verifiers is challenging as verifiers need to capture diverse/realistic behaviors while avoiding adversarial behaviors that no heuristics can handle. CCAC does this by constraining when packets can be delayed/dropped. This requires significant domain expertise and it is unclear if such constraints are strictly necessary. We believe the CEGIS loop can help with tuning verifiers. We can synthesize verifier constraints by asking "$\exists$ constraints on system parameters such that $\forall$ traces that satisfy these constraints, at least one known heuristic achieves its desired goals". The intuition is that different heuristics are designed for different realistic environments. The union of traces over all heuristics captures a broad set of behaviors that realistic systems can exhibit.

ABR. ABR shares similar environments as CCAs, e.g., packet drops, delays, and jitter, but with different objectives, e.g., video quality, playback latency, playback stalls. We were able to reuse CCAC's environment model and encode video quality/stall in terms of playback buffer to build a verifier for ABR. We see this as positive evidence that future work could use automation to distill insights and synthesize robust-by-construction ABR algorithms. Automation could help rapidly specialize offline, live and real-time video streaming. It could further help co-design ABR with congestion control [29], loss recovery mechanisms that mix re-transmission with forward error correction [36], and frame skipping [64].

Scheduling. Scheduling also has a combinatorial explosion in environments (or workloads), objectives, and system interactions. Scheduling decisions depend on factors like preemption and migration overheads, resource constraints, privacy and service-level agreements. As a result, schedulers have been specialized for different workloads and requirements, e.g., data analytics [21, 71], deep learning [33, 43, 51, 68], and short network requests [17, 50, 53]. It is unclear if existing schedulers meet performance bounds. For instance, prior works expose many algorithmic bugs in existing schedulers [42]. Work stealing, to balance load across cores, is a rare exception where we have practically relevant theoretical guarantees [46, 47].

A challenge we anticipate is building an abstraction for the environment. In congestion control/ABR, Network Calculus allowed modeling a variety of sub-RTT phenomena (e.g., ACK aggregation, jitter, token-bucket filters) using a simple packet delay abstraction. We would need a similar way to logically represent environments in scheduling, e.g., placement/locality preferences (data/GPU/NUMA), job/task priorities, communication delays, stragglers.

6 CONCLUSION

We build CCmatic as preliminary evidence for feasibility of modeling and formally reasoning about heuristics in a tractable manner. While we show this in the context of CCA synthesis, we believe our approach can bring clarity to other questions within congestion control and other domains.

Acknowledgements. We would like to thank anonymous reviewers and NSF grants #2212102, and #2212390.

REFERENCES

[1] Alessandro Abate, Cristina David, Pascal Kesseli, Daniel Kroening, and Elizabeth Polgreen. 2018. Counterexample Guided Inductive Synthesis Modulo Theories. In *Computer Aided Verification*, Hana Chockler and Georg Weissenbacher (Eds.). Springer International Publishing, Cham, 270–288.

[2] Susanne Albers, Lene M Favrholdt, and Oliver Giel. 2005. On paging with locality of reference. *J. Comput. System Sci.* 70, 2 (2005), 145–175.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1851182.1851192

[4] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 435–446.

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

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

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

[8] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. 2021. Maximum Satisfiabiliy. In *Handbook of Satisfiability - Second Edition*, Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press, 929–991. https://doi.org/10.3233/FAIA201008

[9] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I (Lecture Notes in Computer Science)*, Dana Fisman and Grigore Rosu (Eds.), Vol. 13243. Springer, 415–442. https://doi.org/10.1007/978-3-030-99524-9_24

[10] Nils Becker, Peter Müller, and Alexander J. Summers. 2019. The Axiom Profiler: Understanding and Debugging SMT Quantifier Instantiations. In *Tools and Algorithms for the Construction and Analysis of Systems: 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Softwawastre, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part I.* Springer-Verlag, Berlin, Heidelberg, 99–116. https://doi.org/10.1007/978-3-030-17462-0_6

[11] Robert D. Blumofe and Charles E. Leiserson. 1999. Scheduling Multithreaded Computations by Work Stealing. *J. ACM* 46, 5 (sep 1999), 720–748. https://doi.org/10.1145/324133.324234

[12] L.S. Brakmo and L.L. Peterson. 1995. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications* 13, 8 (1995), 1465–1480. https://doi.org/10.1109/49.464716

[13] Carlo Caini and Rosario Firrincieli. 2004. TCP Hybla: A TCP Enhancement for Heterogeneous Networks. *Int. J. Satell. Commun. Netw.* 22, 5 (sep 2004), 547–566. https://doi.org/10.1002/sat.799

[14] Neal Cardwell, Yuchung Cheng, Lawrence Brakmo, Matt Mathis, Barath Raghavan, Nandita Dukkipati, Hsiao-keng Jerry Chu, Andreas Terzis, and Tom Herbert. 2013. packetdrill: Scriptable network stack testing, from sockets to packets. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. 213–218.

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

[16] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems* 17, 1 (1989), 1–14. https://doi.org/10.1016/0169-7552(89)90019-6

[17] Inho Cho, Ahmed Saeed, Joshua Fried, Seo Jin Park, Mohammad Alizadeh, and Adam Belay. 2020. Overload Control for {μs-scale} {RPCs} with Breakwater. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 299–314.

[18] Mosharaf Chowdhury and Ion Stoica. 2012. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. 31–36.

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

[20] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2007. Efficient E-Matching for SMT Solvers. In *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings (Lecture Notes in Computer Science)*, Frank Pfenning (Ed.), Vol. 4603. Springer, 183–198. https://doi.org/10.1007/978-3-540-73595-3_13

[21] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[22] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} Vivace:{Online-Learning} Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 343–356.

[23] Nandita Dukkipati and Nick McKeown. 2006. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review* 36, 1 (2006), 59–62.

[24] facebookincubator. 2022. mvfst. https://github.com/facebookincubator/mvfst/blob/94722103c32f11b589f9e0e2165a6b54dbda16d7/quic/congestion_control/Copa2.cpp#L195. (June 2022). https://github.com/facebookincubator/mvfst/blob/94722103c32f11b589f9e0e2165a6b54dbda16d7/quic/congestion_control/Copa2.cpp#L195 [Online; accessed 20. Jun. 2022].

[25] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program Synthesis Using Conflict-Driven Learning. *SIGPLAN Not.* 53, 4 (jun 2018), 420–435. https://doi.org/10.1145/3296979.3192382

[26] Margarida Ferreira, Akshay Narayan, Inês Lynce, Ruben Martins, and Justine Sherry. 2021. Counterfeiting Congestion Control Algorithms. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21)*. Association for Computing Machinery, New York, NY, USA, 132–139. https://doi.org/10.1145/3484266.3487381

[27] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. 2000. Equation-based congestion control for unicast applications. *ACM SIGCOMM Computer Communication Review* 30, 4 (2000), 43–56.

[28] Lance Fortnow. 2022. Fifty years of P vs. NP and the possibility of the impossible. *Commun. ACM* 65, 1 (2022), 76–85. https://doi.org/10.1145/3460351

[29] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 267–282. https://www.usenix.org/conference/nsdi18/presentation/fouladi

[30] Yeting Ge and Leonardo Mendonça de Moura. 2009. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings (Lecture Notes in Computer Science)*, Ahmed Bouajjani and Oded Maler (Eds.), Vol. 5643. Springer, 306–320. https://doi.org/10.1007/978-3-642-02658-4_25

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

[32] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. 2018. Elasticity Detection: A Building Block for Internet Congestion Control. *arXiv* (Feb. 2018). https://doi.org/10.48550/arXiv.1802.08730 arXiv:1802.08730

[33] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 485–500. https://www.usenix.org/conference/nsdi19/presentation/gu

[34] Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. 2011. Synthesis of loop-free programs. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, Mary W. Hall and David A. Padua (Eds.). ACM, 62–73. https://doi.org/10.1145/1993498.1993506

[35] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (jul 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[36] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. 2013. Handling packet loss in WebRTC. In *2013 IEEE International Conference on Image Processing*. IEEE, 1860–1864.

[37] Samuel Jero, Md Endadul Hoque, David R Choffnes, Alan Mislove, and Cristina Nita-Rotaru. 2018. Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach.. In *NDSS*.

[38] Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. 2018. Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rywDjg-RW

[39] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion Control for High Bandwidth-Delay Product Networks. *SIGCOMM Comput. Commun. Rev.* 32, 4 (aug 2002), 89–102. https://doi.org/10.1145/964725.633035

[40] Samuel Kolb, Stefano Teso, Andrea Passerini, and Luc De Raedt. 2018. Learning SMT(LRA) Constraints Using SMT Solvers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. AAAI Press, 2333–2340. https://doi.org/10.24963/ijcai.2018/323

[41] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer.

[42] Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova. 2016. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems*. 1–16.

[43] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 289–304. https://www.usenix.org/conference/nsdi20/presentation/mahajan

[44] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 197–210. https://doi.org/10.1145/3098822.3098843

[45] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. 1997. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Comput. Commun. Rev.* 27, 3 (jul 1997), 67–82. https://doi.org/10.1145/263932.264023

[46] Sarah McClure, Amy Ousterhout, Scott Shenker, and Sylvia Ratnasamy. 2022. Efficient Scheduling Policies for Microsecond-Scale Tasks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1–18. https://www.usenix.org/conference/nsdi22/presentation/mcclure

[47] Sarah McClure, Amy Ousterhout, Scott Shenker, and Sylvia Ratnasamy. 2022. Efficient Scheduling Policies for Microsecond-Scale Tasks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1–18. https://www.usenix.org/conference/nsdi22/presentation/mcclure

[48] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. 2020. PCC proteus: Scavenger transport and beyond. 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*. 615–631.

[49] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. 2018. Restructuring Endpoint Congestion Control. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 30–43. https://doi.org/10.1145/3230543.3230553

[50] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving high {CPU} efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 361–378.

[51] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 3, 14 pages. https://doi.org/10.1145/3190508.3190517

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

[53] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 325–341.

[54] Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. 2022. Moving Definition Variables in Quantified Boolean Formulas. In *Tools and Algorithms for the Construction and Analysis of Systems. 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 462–479. https://doi.org/10.1007/978-3-030-99524-9_26

[55] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. 2015. Counterexample-Guided Quantifier Instantiation for Synthesis in SMT. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Daniel Kroening and Corina S. Pasareanu (Eds.), Vol. 9207. Springer, 198–216. https://doi.org/10.1007/978-3-319-21668-3_12

[56] Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. 2014. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*. IEEE, 195–202. https://doi.org/10.1109/FMCAD.2014.6987613

[57] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark W. Barrett. 2013. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings (Lecture Notes in Computer Science)*, Maria Paola Bonacina (Ed.), Vol. 7898. Springer, 377–391. https://doi.org/10.1007/978-3-642-38574-2_26

[58] Dario Rossi, Claudio Testa, Silvio Valenti, and Luca Muscariello. 2010. LEDBAT: The New BitTorrent Congestion Control Protocol. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks*. 1–6. https://doi.org/10.1109/ICCCN.2010.5560080

[59] Rohit Singh, Rishabh Singh, Zhilei Xu, Rebecca Krosnick, and Armando Solar-Lezama. 2014. Modular Synthesis of Sketches Using Models. In *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings (Lecture Notes in Computer Science)*, Kenneth L. McMillan and Xavier Rival (Eds.), Vol. 8318. Springer, 395–414. https://doi.org/10.1007/978-3-642-54013-4_22

[60] Armando Solar-Lezama. 2021. Lecture 13. https://people.csail.mit.edu/asolar/SynthesisCourse/Lecture13.htm. (April 2021). [Online; accessed 22. Jun. 2022].

[61] Armando Solar-Lezama, Christopher Grant Jones, and Rastislav Bodik. 2008. Sketching Concurrent Data Structures. *SIGPLAN Not.* 43, 6 (jun 2008), 136–148. https://doi.org/10.1145/1379022.1375599

[62] Wei Sun, Lisong Xu, Sebastian Elbaum, and Di Zhao. 2021. Model-Agnostic and Efficient Exploration of Numerical Congestion Control State Space of Real-World TCP Implementations. *IEEE/ACM Transactions on Networking* 29, 5 (2021), 1990–2004.

[63] venkatarun95. 2022. rocc_kernel. https://github.com/venkatarun95/rocc_kernel. (June 2022). https://github.com/venkatarun95/rocc_kernel [Online; accessed 20. Jun. 2022].

[64] Tingfeng Wang, Zili Meng, Mingwei Xu, Rui Han, and Honghao Liu. 2021. Enabling high frame-rate UHD real-time communication with frame-skipping. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*. 19–24.

[65] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of the Internet Measurement Conference (IMC '19)*. Association for Computing Machinery, New York, NY, USA, 137–143. https://doi.org/10.1145/3355369.3355604

[66] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-Generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 123–134. https://doi.org/10.1145/2486001.2486020

[67] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, Lombard, IL, 459–471. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/winstein

[68] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[69] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 495–511. https://www.usenix.org/conference/nsdi20/presentation/yan

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

[71] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 15–28.

[72] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 509–522. https://doi.org/10.1145/2785956.2787498

[73] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484