NetReplica: A Bottleneck-Regime Substrate for Reproducible Network Data Generation

Jaber Daneshamooz
UC Santa Barbara
Santa Barbara, California, USA

Satyandra Guthula
UC Santa Barbara
Santa Barbara, California, USA

Jessica Nguyen
UC Santa Barbara
Santa Barbara, California, USA

William Chen
Duke University
Durham, North Carolina, USA

Sanjay Chandrasekaran
UC Santa Barbara
Santa Barbara, California, USA

Ankit Gupta
AMD
Santa Clara, California, USA

Arpit Gupta
UC Santa Barbara
Santa Barbara, California, USA

Walter Willinger
Northwestern University
Evanston, Illinois, USA

Abstract

[WW: An application's performance is largely determined by the conditions at its bottleneck link, yet experiments designed to study this performance are typically described in terms of the testbeds and tools that produce those conditions, not by the conditions themselves. As a result, each study specifies its conditions in its own testbed-specific way, making them difficult to reproduce, port, or verify. In an audit of 66 measurement papers, 255 of 387 experiments (66%) required that rather than replaying existing traffic or simulating or synthesizing it offline, the relevant network condition had to be generated "by hand" in a time-consuming manner. To accelerate this manual effort, this paper presents NetReplica, a bottleneck-regime substrate that consists of a grammar that allows for the precise specification of a bottleneck's static envelope and dynamic congestion pressure and an execution plane that instantiates the resulting regime independent of where or how an experiment runs. Through progressive disaggregation, applied three times, NetReplica makes a regime a first-class, composable, replayable object: a bottleneck link specification that is precise and shared (i.e., every flow through the bottleneck link experiences the same regime) and per-flow specifications that approximate the rest of each flow's path that traverses the bottleneck link. To demonstrate the potential of NetReplica, we regenerated a published measurement result end to end: a deployed congestion-control classifier's per-algorithm queue-occupancy signatures, for all 15 classified algorithms, from a regime specification that is independent of the classifier's original testbed. NetReplica supplies these regime specifications to an empirical research backend that produces trustworthy, never-synthetic evidence, composed only from observed traces rather than a learned model, an achievement no single existing system can currently accomplish. We state precisely where the design specifies more than the current implementation of NetReplica delivers.]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
NTC-Reloaded '26, Utrecht, The Netherlands
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

CCS Concepts

• **Networks** → **Network experimentation**; *Network measurement*; *Network performance evaluation.*

Keywords

reproducibility, network experimentation, bottleneck regime, cross-traffic, network data generation

ACM Reference Format:
Jaber Daneshamooz, Satyandra Guthula, Jessica Nguyen, William Chen, Sanjay Chandrasekaran, Ankit Gupta, Arpit Gupta, and Walter Willinger. 2026. NetReplica: A Bottleneck-Regime Substrate for Reproducible Network Data Generation. In *Proceedings of NTC-Reloaded '26, co-located with ACM CoNEXT 2026 (NTC-Reloaded '26).* ACM, New York, NY, USA, 8 pages.

1 Introduction

An application's performance over the Internet is to a large degree governed by one constraint: the active bottleneck link on its path. That link's static properties that include capacity, base latency, and queue configuration set an operating envelope, but the envelope alone does not determine what

an application experiences. Under the same envelope, different dynamic congestion pressure (i.e., how much competing traffic there is, how bursty it is, when it arrives) produces different queueing, loss, and delay, and therefore different application behaviors. We call this pair of a static envelope and the dynamic pressure contending within it a *bottleneck regime*.[1] Reproducing an application's behavior or comparing two systems fairly therefore means reproducing the full regime it ran under, not just the envelope.

Generating data under a chosen regime is challenging because experiments run at the level of executions, not regimes. To run an experiment, instead of first specifying it and then realizing it, a researcher typically chooses an execution context (e.g., which testbed or cloud to use, which type of traffic or trace to consider, which transport stack to leverage), and that choice usually fixes the regime implicitly. As a result, precisely stating the same intended regime, varying it along one axis while holding others fixed, or reinstating it in a different environment become challenging tasks and require significant efforts. This is not a rare inconvenience: we manually audited 387 experiments drawn from 66 recent measurement papers, and 255 of them (66%) require generating network conditions end to end and endogenously; running the remaining experiments requires either replaying preexisting traffic, running a simulator, or performing offline computations [8]. Each of these 255 experiments depends on a bottleneck regime, specified explicitly or left implicit in a testbed's default configuration, and each paper uses whatever type of traffic, ad hoc packet-capture replay, or public dataset is convenient and available; none of the papers reuses another paper's regime. This is a reinvention problem one layer below the one our community usually discusses: the problem is not which model architecture or training dataset but which network conditions governed an experiment in the first place.

These observations motivate four requirements for any proposed bottleneck-regime specification. It must let users *(i)* specify static attributes and dynamic back pressure independently of the execution context (*controllability*); *(ii)* combine independently specified static attributes and dynamic pressure predictably into regimes never jointly observed in any one measurement (*composability*); *(iii)* preserve real, closed-loop interactions between the application under test and the contending traffic at the bottleneck while accurately realizing the specified pressure (*fidelity*); and *(iv)* reinstate the same specification to obtain statistically equivalent behavior across runs and environments (*replicability*).

No existing approach satisfies all four requirements together; each covers only some subset. For example, public traces [10, 19] are realistic but implicit. They do not expose independent controls for capacity, cross-traffic, and latency inside a trace someone else collected, and as CausalSim [1] documents for the case of collecting data to examine an ABR algorithm's own influence on Puffer's traces, the traces themselves inherit whatever bias their original collection introduced. Record-and-replay [14, 16, 18], on the other hand, limits decomposability. While it preserves a captured combination faithfully, it does not itself provide a way to recombine its components across new contexts. In contrast, programmable emulation testbeds and congestion-control benchmarking platforms [12, 13, 20] allow for precise execution control, but the regime itself (e.g., choice of cross-traffic, selection of capacity trajectory) requires manual specification, one study at a time, resulting in the above-mentioned reinvention problem. For instance, the authors of Pantheon [20] note that their fixed, heuristically chosen set of bottleneck attributes may prove insufficient as new congestion-control schemes get added, exactly the extensibility gap that a a bottleneck-regime substrate with a shared grammar would close. Yet another approach that addresses an entirely different problem concerns learned traffic generators [15] that sample from a statistical model of network traffic fit to some corpus and trade fidelity for composability: a model evaluated against such sampled traffic risks matching the generator's inductive biases rather than the network dynamics the study intends to test. We return to this comparison in Section 6.

NetReplica is an approach that satisfies all four requirements simultaneously: a bottleneck-regime substrate that pairs a grammar for regimes with the execution plane that instantiates them. NetReplica makes a regime a first-class, specifiable, composable, and replayable object, independent of where or how it executes (Sections 3–4). To illustrate, we use NetReplica to reproduce a published measurement result end to end: the per-algorithm queue-occupancy signatures reported by a deployed congestion-control classifier [17], for all 15 classified algorithms, regenerated from a specification independent of the classifier's original testbed (Section 5). We reproduce this one experiment in detail and leave an in-depth evaluation of the classifier's downstream performance for future work. This example shows that instead of inheriting a regime implicitly from whatever testbed or trace a paper happened to use and then trying to reinstantiate it under different conditions, all that is required from a researcher is the one-time effort to specify, port, and check the regime under which an experiment or study was run or performed. NetReplica supplies these regime specifications to an empirical research backend that promises trustworthy, never-synthetic evidence for research intents [8]. More concretely, a system like NetReplica is need to make good on

[1] We write "regime" alone as shorthand for "bottleneck regime" throughout; no other kind of regime appears in this paper.

this promise for the type of condition-driven experiments described in Section 2.

This paper evaluates NetReplica on its own terms and is a shorter version of a technical report [6]. Section 2 provides a fuller account of the reinvention argument; Section 3 presents NetReplica's design; Section 4 addresses how conditions stay faithful to production behavior; Section 5 reports the reproduction result; Section 6 positions NetReplica against related approaches; and Section 7 concludes with a discussion of where NetReplica's design is ahead of its implementation.

2 Motivation

We obtained Section 1's 66% figure by first hand-selecting a set of replication-worthy measurement papers from the existing literature and then using their ACM CCS concepts and author-provided keywords as a search vocabulary. Next, we collected a total of some 146 candidate papers that appeared in the last 10 years, were published in SIGCOMM, CoNEXT, IMC, or SIGMETRICS, and whose concepts, keywords, title, or abstract resulted in matches with this vocabulary. A manual screening to these paper for replicability left us with a total of 66 papers which contained specific experiments, where an experiment is defined in terms of a distinct setup rather than a figure. For each of these 66 papers, we reconstructed its experiments, either from the paper's figures, tables, captions, or in-text references. We recorded a setup value only when the paper states it explicitly and marked it as not specified otherwise. Finally, to simplify the decision making about whether an experiment requires generating a precisely-defined network condition, we encoded each experiment using a closed grammar for bottleneck conditions, network paths properties, congestion control, and queues [8]. As shown in Table 1, the 66 audited papers span a broad range of networking topics, with video streaming and congestion control account for 29 of the papers and 204 of the identified experiments.

This 66% number is a symptom of what ails reproducible network data generation. This section names the disease, while Section 3 offers a cure in the form of a design of a novel bottleneck-regime substrate. To describe the disease, consider a classifier, foundation model, or protocol evaluated against network conditions that the team of paper authors chose for their paper and that therefore inherits whatever bias that one-time choice encodes. A second team attempting to check or extend the result can reproduce the reported *numbers* but there is in general no way for this team to confirm that they succeeded in reproducing the relevant network conditions: testbed particulars are usually described in prose and not specified precisely enough to instantiate them elsewhere. This is a reproducibility problem in a mechanical sense; it differs from trained models exploiting shortcuts encoded in their training data [5, 9], and from an entire evaluation being compromised because of unexamined choices of realistic network conditions.

Existing approaches offer no cure for this disease but instead solve adjacent problems. While present-day testbeds allow for precise execution control, they require the regime itself (e.g., type of cross-traffic, evolution of capacity) to be hand-specified per study. Similarly, while datasets and traces guarantee realism, they couple it to the context that produced them, and researchers cannot vary a regime observed once or recombine it with another regime. What is missing is a shared representation for a bottleneck regime, independent of any single paper's testbed or trace: something researchers can specify once, then reuse, recombine, or extend; something that can be checked against production behavior and doesn't have to be rebuilt ad hoc every time.

3 Design and Implementation

NetReplica resolves Section 2's reinvention problem with the same systems principle that we call *progressive disaggregation*. The same way protocol layering or control-data plane separation resolved analogous couplings elsewhere in networked systems, we use this principle here and apply it three times: separating the regime from the execution context, separating static attributes from dynamic pressure, and separating a trace's dynamics from where it was captured. Figure 1 shows three distinct layers that illustrate these three disaggregation steps, namely (top to bottom): trace-context, static-dynamic, and intent-execution disaggregation.

Users specify the bottleneck link itself precisely, and this specification is shared across every contending workflow through the bottleneck link; NetReplica approximates the rest of each such workflow's path, a consequence of the second separation applied to the workflow-specific aspects of the bottleneck, not a fourth separation. Section 1 defined

Table 1: The 66 audited papers by main area (each paper counted once, under its primary area), with the number of experiments extracted from them.

Area	Papers	Exp.
Video streaming and QoE	18	100
Congestion control	11	104
Measurement, wireless, DNS, infrastructure	9	51
Transport and multipath	7	50
Queue management and data plane	6	44
ML, emulation, user studies	6	16
Real-time and interactive apps.	5	31
Web performance	4	14
Total	66	410

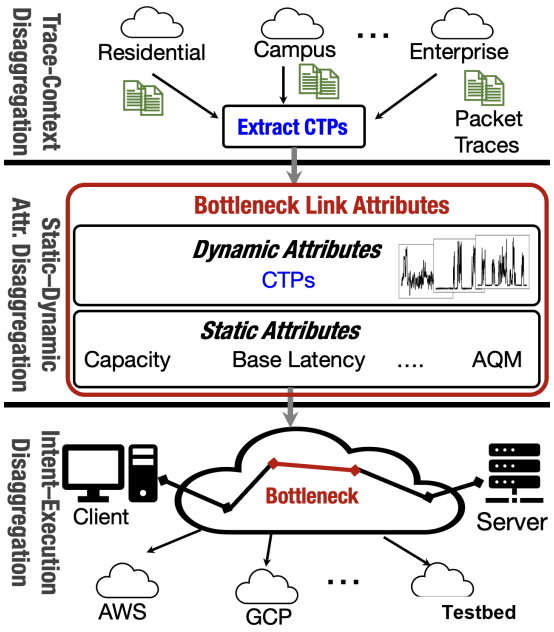


Figure 1: NetReplica's three disaggregation layers: trace-context, static-dynamic, and intent-execution.

a bottleneck regime informally as a static envelope paired with the dynamic pressure contending within it. For such a pair that concerns the one dominant link that constrains the experiments NetReplica targets (i.e., the bottleneck, the main object of our study), NetReplica's grammar makes this definition formal and is described below:

```
Regime = Static<capacity(down,up), queue> ⊗ Dynamic<pressure>
queue = <discipline, size, params, ecn, mode>
pressure = replay(ctp) | load(workflow) -> node() | both
```

`Static` fixes the envelope: link capacity and the queue that builds against it. `Dynamic` fixes the pressure: competing traffic contending within that envelope. And while `replay` draws on a reusable Cross-Traffic Profile, or `ctp`, defined below, `load` generates that traffic from co-scheduled workflows, detailed in Section 4.

Spec versus execution. Users specify a regime independently of the environment that will realize it (Figure 1's intent-execution layer at the bottom): a local container, a cloud instance, a physical testbed. One specification serves multiple substrates; changing where an experiment runs never changes what it specifies, which is what makes the specification replicable across both runs and environments.

The bottleneck (precisely defined) versus the traffic through the bottleneck (approximately specified). This `Static/Dynamic` split is Figure 1's static-dynamic layer in the middle. Within `Static`, the queue is a structured object, not a flat selector, because studies that vary an active queue management (AQM) discipline, buffer size, or explicit congestion notification (ECN) marking require that each be a typed field, not an opaque string. Every workflow contending through the bottleneck experiences the same regime, so this specification is not per-workflow but shared. Congestion delay and loss at the bottleneck are not knobs: queue together with `pressure` determine these properties, so the regime sets the link and the load, and the resulting queueing is the phenomenon under study. NetReplica approximates the rest of each workflow's path to that bottleneck with three workflow-dependent parameters (i.e., latency, jitter, and loss); capacity is deliberately absent here, because only the bottleneck constrains it, and two workflows sharing this one bottleneck can still traverse otherwise different routes with different latencies. This split separates a shared, precisely defined component of the object of interest (i.e., the bottleneck link) from approximate, per-workflow components for everything else on the workflows' paths and realizes NetReplica's central architectural claim: letting researchers vary a regime specified for one study or recombine it with another regime and using such obtained regimes for new studies.

One bottleneck but multiple contending workflows. A regime's realized queueing depends on not just one workflow but on all workflows that contend at the bottleneck: two contending workflows running different congestion-control algorithms produce different queueing even under the same static envelope and the same background pressure. NetReplica's grammar therefore references a foreground of externally specified workflows placed on nodes, `Foreground = {Workflow_i @ node_i}`, so that a target application and one or more contending workflows share the same bottleneck within a single specification. NetReplica does not author or specify what a workflow does, it only has knowledge of how the workflow is situated relative to the bottleneck. [JD: A *workflow* is a specified application or traffic-generating activity in the network (e.g., watching a video on YouTube or running iperf3). A *contending workflow* is any workflow that shares the same bottleneck with the target application, also called the *workflow under test* (or application under test).] The workflows themselves can be automated with agent-based tools such as NetGent [7]. For instance, a video-streaming workflow contending against a bulk-transfer workflow over a given specified regime is one `Foreground`, not two separate experiments that have been stitched together after runnning them separately. NetReplica tracks each contending workflow's congestion-control configuration as a structured object, `cca = <algo, stack, version, params, flags, build>`. This is necessary because the same algorithm behaves differently across implementations (e.g., a kernel module against a userspace QUIC

stack, or one kernel build against another), and this difference changes the regime the workflow under test actually experiences. Composability is critical for this to work correctly: capacity, queue, cross-traffic pressure, and each workflow's path and congestion-control configuration combine independently, so a regime built for one study's contention pattern recombines into a new one without re-deriving the underlying bottleneck specification.

CTPs make congestion pressure reusable. The third separation, Figure 1's trace-context layer at the top, frees a trace's dynamics from the context in which NetReplica captured it. Cross-Traffic Profiles (CTPs) express that separated pressure: a passively collected packet-level trace from a vantage point within a production network becomes a reusable pressure program decoupled from the network on which NetReplica captured it. Turning a trace into CTPs is not a raw replay: NetReplica trims a capture to its congestion-relevant window, scales its rate to a target capacity, and recombines segments from different captures into a single dynamic congestion pressure program, so a regime can compose bursts that never co-occurred in any one production trace. The burstiness characterization that guides which segments to select as dynamic congestion pressure programs leverages recent efforts on event-centric modeling of bursty traffic time series [11]. This approach does not result in "invented" traffic: each recombined segment is itself an observed capture, so the obtained evidence remains never-synthetic even when the combination is not. NetReplica's current prototype builds its CTP corpus from a campus network gateway trace; CTP extraction only assumes timestamped packet arrivals and sizes, so the method ports to other vantage points, and the campus-trace choice is a deployment artifact, not an architectural constraint. We flag this choice rather than hide it: prior campus-network measurement work has found that campus infrastructure is typically over-provisioned relative to last-mile access conditions [3], so broadening the CTP corpus beyond a single campus vantage point is a concrete next step as part of our future work.

Checked fidelity. The separations resulting from the three applications of the disaggregation principle illustrated in Figure 1 make a regime specifiable. To check whether it is also trustworthy, rather than simply assuming that an instantiated regime holds, NetReplica checks whether a regime is instantiated correctly. To this end, NetReplica probes the realized capacity, latency, and queueing before an experiment's data capture begins, and records the realized-versus-requested conditions alongside every result, so a downstream consumer of NetReplica-generated data can check for any discrepancies between the specified and realized conditions rather than naively trusting that the specified conditions have been instantiated correctly. The grammar names two verification modes: *imposed*, where a probe confirms the specified regime before the target workflow starts, and *inhabited*, where the probe instead characterizes whatever regime is already present. The latter mode is useful when a study wants to know what conditions exist rather than assert what they should be.

4 Closed-Loop Fidelity

NetReplica's approach to network data generation raises the important question whether replaying a captured cross-traffic pattern that is unresponsive to the bottleneck it now shares with a workflow under test can produce faithful stress conditions. The answer to this question is that NetReplica's hybrid design emphasizes a deliberate compromise. Background pressure specified through a `replay(ctp)` is open loop: the pressure results from playing back a fixed, previously captured pattern regardless of how the bottleneck responds. That choice is deliberate. Reactive replay would distort the burst structure and temporal correlations a CTP preserves. The design assumes that at realistic link occupancies, one workflow's contribution to aggregate congestion is not significant enough to be treated as exogenous, so open-loop background pressure aims to model that regime faithfully rather than simplify it. In the case of the classifier setting considered in Section 5 below, we show that the reproduced network traffic is consistent with that assumption. Prior work has questioned this exogenous assumption. For example, iBox instead learns a reactive cross-traffic model via Bayesian optimization [2], trading NetReplica's portability (NetReplica mines a CTP once, then replays it anywhere) for tighter fit to one measured path. NetReplica's and iBox's approaches complement each other: `load` mode, discussed below, is NetReplica's own way toward the same reactive cross-traffic capability. [JD: The *workflow under test* (equivalently, the application under test or target application) is the workflow whose behavior the experiment studies; it is always endogenous and closed-loop.] The workflow under test differs from the background pressure under every mode: an unmodified application and transport stack executes over a real, verified queue, so every interaction between that workflow under test, its transport, and the bottleneck is real rather than modeled. We call this "closed-loop" to express the fact that rather than simply replaying an offline trace, the workflow under test reacts to the bottleneck's current state the same way it would in a live deployment. As a substrate, NetReplica therefore behaves like a network emulator rather than a simulator: it specifies background pressure in an open-loop fashion through `replay(ctp)`, and the workflow under test always experiences that pressure in a closed-loop fashion.

This open-loop/closed-loop split is a specific case of a broader distinction the networking community already names: traffic is either *endogenous*, generated from within the system under study and reactive to it, or *exogenous*, imposed from outside and indifferent to the system under study. While `replay(ctp)` pressure is exogenous by design, the workflow under test is always endogenous. The grammar's `pressure` production already names a second mode beyond replay, `load(workflow(*)` → `node(*))`, in which the execution plane co-schedules live workflows as the cross-traffic itself, making the pressure endogenous and reactive rather than exogenous and replayed. Section 7 discusses this mode's implementation status.

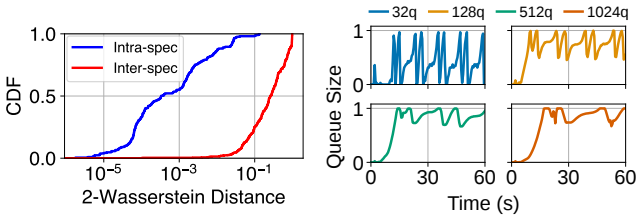
5 Evaluation

We validated each of the four requirements detailed in Section 1 using the current NetReplica prototype, and the first three reported results summarize this validation effort. We then transferred one result to an external, published benchmark, and the last result described in this section concerns this transferred reproduction, the paper's primary new evidence.

Controllability and composability. When sweeping capacity (10, 100, 1000 Mbps) at fixed base RTT values, measured throughput increases monotonically while minimum RTT stays invariant; similarly, when sweeping base RTT (10, 100, 1000 ms) at fixed capacities, measured RTT increases while throughput stays invariant. Neither of the sweep leaks into the other's observable. Holding capacity, base RTT, and queue discipline fixed while swapping between two CTPs produces distinct, CTP-dependent throughput and RTT dynamics; holding a CTP fixed while swapping the queue discipline among FIFO, CoDel, and FQ-CoDel produces distinct, discipline-specific delay behaviors, withe FIFO inflating delay under load and CoDel controlling it. Each attribute's effect surfaces on its own, independent of which other attributes are also specified.

Replicability. Across ten independent instantiations per specification, we compute the pairwise 2-Wasserstein distance between different RTT and throughput time series, respectively. Figure 2a shows that the inter-specification distances are tightly concentrated around a small mean while the intra-specification distances are much more spread out and consistently smaller than their inter-specification counterparts: the observed variability traces to the specification, not to execution noise and suggests that this variability is dominated by specification differences rather than execution instability.

Fidelity, quantitatively. Beyond the closed-loop argument in Section 4, dynamic-pressure fidelity holds statistically as well. Across 100 samples of same common mean offered load



(a) CDF **(b) Replicate**

Figure 2: (a) Replicability under repeated instantiation: intra- vs. inter-specification 2-WD distance distributions across all the sweeps, (b) Replicating the experiments corresponding to Figure 5 of [17]: We validated these results with the authors of [17] who attributed subtle discrepancies in the queue occupancy tie series to minor differenes in measuring queue occupancy.

CTPs, the coefficient of variation of the specified pressure correlates with the coefficient of variation of the resulting application throughput (Pearson $r = 0.33$, $p = 0.0003$): more bursty pressure produces more bursty observed throughput. The same correlation does not hold for RTT ($r = -0.03$, $p = 0.72$), consistent with the target application's own bufferbloat masking exogenous pressure once the bottleneck saturates, not with a failure to preserve dynamic structure.

A transferred reproduction. So far, when evaluating our system, we specified the regime for each run (e.g., capacity, queue discipline, competing traffic pattern) in NetReplica's grammar and instantiated it on infrastructure that is independent of that of a deployed congestion-control classifier's own original testbed [17]. That specification alone regenerated the queue-occupancy time series the classifier's central experiment reports, across all 15 congestion-control algorithms the classifier distinguishes, producing 15 distinct, correctly separated queue signatures over 20 out of 20 runs. A small sample of results for replicating one of [17]'s experiments (i.e., reproducing [17]'s Figure 5 by sweeping the queue size) is shown in Figure 2b(b). To scope this result more precisely: we reproduce the queue-occupancy signatures the classifier consumes, and we leave the evaluation of its downstream classification accuracy and its remaining reported results for future work. This reproduction bears directly on a reproducibility-focused venue: a bottleneck regime specified once, independent of the original study's testbed, regenerated the empirical basis of a separately published finding, the property Section 2 argues today's ad hoc, hand-built condition specifications cannot offer.

6 Related Work

NetReplica compares against learned, synthetic traffic generators, the fourth class mentioned in Section 1, and against

the three approaches already surveyed there. NetReplica deliberately takes a position that is opposite from that taken by generators: every dynamic pressure signal NetReplica composes comes from a real, captured trace, and every workflow it drives is a real, unmodified application, composed under controlled conditions instead of sampled from a learned model. Emulation testbeds [12, 13] and record-and-replay systems [14, 16, 18] established the value of real applications over real, monitored network elements; NetReplica keeps that commitment and adds the piece those systems leave hand-specified or trace-bound, the regime itself, as a first-class, composable object. Platforms for portable, heterogeneous experiment execution [4] solve a complementary problem, namely placing an experiment across diverse infrastructure. NetReplica's regime specification composes with that execution layer instead of duplicating it. No prior approach combines independent control over the regime, composable recombination, fidelity to real dynamics, and replicable behavior once specified; each covers some subset, exactly the gap Section 2's 66% figure shows still needs closing.

7 Discussion and Conclusion

Design ahead of implementation. NetReplica's closed-loop fidelity claim (Section 4) covers realism for the workflow under test against a checked queue, delivered today; it does not yet cover fully reactive background pressure, and two parts of NetReplica's design specify more than the current prototype delivers. First, CTP selection and representation is coarser than the grammar's fidelity goal: the prototype characterizes profiles by variance-based clustering, and a richer, embedding-based characterization is the identified next step, not yet built. Second, the `load` pressure mode from Section 4 lets the execution plane co-schedule live workflows as reactive, endogenous cross-traffic, extending `replay(ctp)`'s exogenous pressure rather than replacing it. This mode is a named grammar production whose full co-scheduling implementation is not yet finished; the current prototype already exposes the abstraction boundary a co-scheduling implementation needs, but the implementation itself remains incomplete [7]. A third, related direction is generalization to regimes with multiple simultaneous or migrating bottlenecks (but see [6] for some initial efforts).

Conclusion. NetReplica regenerated a published measurement result end to end from a specification independent of its original testbed. We take this as evidence that a bottleneck regime, once formalized, transfers cleanly across execution environments rather than staying bound to the one that produced it. That transfer works because NetReplica's grammar turns a regime into a shared, composable, checkable, and real, never-synthetic specification, through the same principle, progressive disaggregation, applied three times. Some 66% of the measurement-paper experiments we audited depend on exactly this kind of specification: the network condition an experiment runs under, not its application workflow, determines the outcome, and today authors reinvent that condition by hand in every paper, or, increasingly, a model trained to imitate it synthesizes it instead. An application's bottleneck was always going to govern its performance; what remains is finishing what the grammar already specifies for that bottleneck: richer cross-traffic characterization and fully reactive cross-traffic pressure, both named, scoped, and in progress.

References

[1] Abdullah Alomar, Pouya Hamadanian, Arash Nasr-Esfahany, Anish Agarwal, Mohammad Alizadeh, and Devavrat Shah. 2023. CausalSim: A Causal Framework for Unbiased Trace-Driven Simulation. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*. USENIX Association.

[2] Sachin Ashok, Shubham Tiwari, Nagarajan Natarajan, Venkata N. Padmanabhan, and Sundararajan Sellamanickam. 2022. Data-Driven Network Path Simulation with iBox. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)* 6, 1 (2022). https://doi.org/10.1145/3508026

[3] Roman Beltiukov, Sanjay Chandrasekaran, Arpit Gupta, and Walter Willinger. 2023. PINOT: Programmable Infrastructure for Networking. In *Proceedings of the 2023 Applied Networking Research Workshop (ANRW '23)*. ACM. https://doi.org/10.1145/3606464.3606485

[4] Roman Beltiukov, Wenbo Guo, Arpit Gupta, and Walter Willinger. 2023. In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. ACM. https://doi.org/10.1145/3576915.3623075

[5] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. 2022. Underspecification Presents Challenges for Credibility in Modern Machine Learning. *Journal of Machine Learning Research* 23, 226 (2022), 1–61.

[6] Jaber Daneshamooz, Satyandra Guthula, Jessica Nguyen, William Chen, Sanjay Chandrasekaran, Ankit Gupta, Arpit Gupta, and Walter Willinger. 2026. NetReplica: A Programmable Substrate for Bottleneck-Centric Network Data Generation. (2026). arXiv:cs.NI/2507.13476 https://arxiv.org/abs/2507.13476

[7] Jaber Daneshamooz, Eugene Vuong, Laasya Koduru, Sanjay Chandrasekaran, and Arpit Gupta. 2025. NetGent: Agent-Based Automation of Network Application Workflows. *arXiv preprint arXiv:2509.00625* (2025).

[8] Jaber Daneshamooz, Eugene Vuong, Alagappan Ramanathan, Manni Moghimi, Haarika Manda, Satyam Kumar, Snithik Thode, Satyandra Guthula, Sylee Beltiukov, Dongsu Han, et al. 2026. Pramana: A Composable, Domain-Specific Backend for Empirical Networking Research.

arXiv preprint arXiv:2607.26352 (2026). To appear in Proceedings of HotNets 2026.

[9] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. 2020. Shortcut Learning in Deep Neural Networks. *Nature Machine Intelligence* 2 (2020). https://doi.org/10.1038/s42256-020-00257-z

[10] Phillipa Gill, Christophe Diot, Lai Yi Ohlsen, Matt Mathis, and Stephen Soltesz. 2022. M-Lab: User Initiated Internet Data for the Research Community. *ACM SIGCOMM Computer Communication Review* 52, 1 (2022). https://doi.org/10.1145/3523230.3523236

[11] Satyandra Guthula, Jaber Daneshamooz, Charles Fleming, Kesheng Wu, Walter Willinger, and Arpit Gupta. 2025. Netburst: Event-centric forecasting of bursty, intermittent time series. *arXiv preprint arXiv:2510.22397* (2025).

[12] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible Network Experiments Using Container-Based Emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. ACM. https://doi.org/10.1145/2413176.2413206

[13] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IX)*. ACM. https://doi.org/10.1145/1868447.1868466

[14] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC '15)*. USENIX Association.

[15] Santosh Kumar Nukavarapu, Mohammed Ayyat, and Tamer Nadeem. 2022. MirageNet: Towards a GAN-Based Framework for Synthetic Network Traffic Generation. In *Proceedings of the 2022 IEEE Global Communications Conference (GLOBECOM '22)*. IEEE, 3089–3095.

[16] William Sentosa, Balakrishnan Chandrasekaran, P. Brighten Godfrey, and Haitham Hassanieh. 2025. CellReplay: Towards Accurate Record-and-Replay for Cellular Networks. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '25)*. USENIX Association.

[17] Ranysha Ware, Adithya Abraham Philip, Nicholas Hungria, Yash Kothari, Justine Sherry, and Srinivasan Seshan. 2024. CCAnalyzer: An Efficient and Nearly-Passive Congestion Control Classifier. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM. https://doi.org/10.1145/3651890.3672255

[18] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*. USENIX Association.

[19] 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 *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*. USENIX Association.

[20] Francis Y. Yan, Jestin Ma, Greg Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: The Training Ground for Internet Congestion-Control Research. In *Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC '18)*. USENIX Association.