R-Hybrid: Fast Receiver-Driven Congestion Regime Detection for Adaptive Congestion Control Switching in AWS

Khuong Nguyen
University of Virginia
Charlottesville, VA, USA

ABSTRACT

Modern cloud networks often expose multiple congestion regimes along a single end-to-end path, ranging from isolated low-latency intra–Availability Zone segments to heavily shared cross-AZ and Internet bottlenecks. No single congestion control algorithm performs optimally across these heterogeneous environments. Receiver-driven designs such as DCTCP and Homa excel on exclusive links, while sender-driven WAN algorithms like BBR or PCC are more robust under contention. However, public cloud tenants lack in-network visibility or control, making existing hybrid schemes difficult to deploy. R-Hybrid addresses this gap by introducing a lightweight, endpoint-only mechanism for fast congestion regime detection and adaptive congestion control switching. A receiver-side agent classifies the current regime using RTT tail, ECN, and loss signals, then instructs the sender to switch between existing Linux CC modules. We evaluate R-Hybrid on AWS across multiple path types and traffic patterns, showing that it achieves low latency on private paths and fairness under shared bottlenecks, with minimal overhead and no network infrastructure changes.

1 INTRODUCTION

Modern cloud networks often encompass two very different congestion regimes within a single flow's path: (1) segments where the flow is effectively isolated (e.g. within a single tenant's environment or an intra-Availability Zone link), and (2) segments that are shared bottlenecks (e.g. cross-AZ or Internet egress points where many flows compete). Existing congestion control (CC) algorithms tend to be optimized for one scenario or the other, leading to performance problems when misapplied. For example, Data Center TCP (DCTCP) relies on shallow ECN marking in switches to keep queues short while sustaining high throughput, but it is intended only for single-admin domains (not safe over the general Internet without modifications) [3]. Conversely, a WAN-optimized CC like BBR tries to operate at the optimal bandwidth-delay product with near-empty queues by capping in-flight data to the pipe's BDP [4], which can underutilize a dedicated high-speed link in a data center setting. This mismatch can cause either excessive queueing (bufferbloat) or poor throughput and link underutilization.

In public cloud environments (such as AWS), these issues are aggravated by the opaque network constraints and limited visibility. Cloud tenants only control the endpoints (VMs) and typically cannot rely on custom switch configurations, in-network telemetry, or even consistent ECN support on all paths. In fact, public cloud networks impose their own limits and behaviors – for instance, a single AWS Internet Gateway (IGW) may cap multi-flow egress bandwidth and NAT gateways start at 5 Gbps and scale up to 100 Gbps on demand [2]. Moreover, traffic between Availability Zones (AZs) is routed through distinct failure domains (separate AZ fabrics), adding latency and variability [2]. These cloud-specific factors mean a transport algorithm cannot assume homogeneous, well-known network behavior; it must adapt dynamically using only end-to-end observations.

This leads to a fundamental challenge: a cloud flow may encounter multiple congestion regimes that demand fundamentally different congestion-control strategies, yet the sender has no information about where regime boundaries lie or when they change. A single, static congestion control algorithm cannot accommodate both single-owner datacenter behavior and shared WAN bottleneck behavior. What is needed is a mechanism that can detect, at runtime and using only endpoint observations, whether a flow currently behaves like it is in a private, low-latency regime or a contended, multi-tenant regime—and adapt its congestion control strategy accordingly.

This paper introduces R-Hybrid, an endpoint-only hybrid congestion control architecture designed to address this challenge in unmodified public cloud networks. R-Hybrid is based on a key insight: although the sender has limited visibility into congestion conditions, the receiver directly observes forward-path signals that differentiate the two regimes. On an isolated cloud link, the minimum RTT remains stable, the RTT tail stays close to baseline, and ECN or loss events are rare. Under shared bottlenecks, persistent queuing inflates RTT tails, ECN congestion-experienced marks accumulate when supported, and retransmissions become more frequent.

R-Hybrid exploits this asymmetry by running a lightweight receiver-side agent that continuously samples these signals, classifies the current regime using temporal smoothing and hysteresis, and instructs the sender to switch between appropriate Linux CC modules (e.g., DCTCP for single-owner links, CUBIC/BBR for shared ones).

R-Hybrid requires no changes to AWS infrastructure, NIC firmware, or switch configurations. All logic resides in userspace at the receiver and uses standard kernel interfaces at the sender, making the system deployable on ordinary EC2 instances. The design incorporates windowed smoothing, hysteresis, and cooldown controls to avoid oscillation and ensure stable behavior despite noisy measurements.

This project makes four contributions. First, it identifies the problem of congestion regime ambiguity in public cloud environments and shows why no single congestion control algorithm can perform well across these heterogeneous conditions. Second, it introduces R-Hybrid, a practical receiver-driven hybrid congestion control architecture that operates entirely at the endpoints and requires no support from the underlying network. Third, it proposes a simple yet robust regime-detection algorithm that combines RTT tail behavior, ECN activity, and retransmission signals, along with stability mechanisms that prevent oscillation between modes. Finally, it evaluates R-Hybrid on AWS EC2 across intra-AZ, cross-AZ, and Internet paths, demonstrating that the system achieves low latency on single-owner links and fairness under shared bottlenecks, outperforming static baselines such as CUBIC, BBR, and DCTCP when these algorithms are used in regimes for which they were not designed. The full implementation of R-Hybrid, including the sender and receiver agents, experimental scripts, and data-processing tools, is available as open-source at: `https://github.com/knguyen2000/r-hybrid`.

2 BACKGROUND AND MOTIVATION

The performance a transport protocol achieves in the cloud depends heavily on the congestion regime along its path. Prior work shows that receiver-managed or ECN-based protocols can maintain extremely low queueing delay and high utilization in private, single-tenant environments [9], while classical sender-driven loss or delay-based approaches are necessary for achieving fairness on shared bottlenecks [4]. Unfortunately, cloud paths often include both types of segments, and no single congestion-control algorithm performs well across them.

If a flow could correctly infer which regime currently dominates its bottleneck, it could adopt the CC mode best suited to that environment. This would allow applications to maintain low tail latency on private links while preserving fairness and stability when competing across AZ boundaries or Internet egress points. Such adaptability would directly benefit cloud applications that are sensitive to both throughput and latency, including RPC frameworks, microservices, and interactive data-processing systems. The design challenge is therefore not simply selecting a better CC algorithm, but enabling mode switching at the right time using only endpoint-visible signals. An effective solution must respond rapidly to regime changes, avoid misclassification that causes oscillation, and operate entirely without in-network support-constraints imposed by today's public cloud platforms.

3 PROJECT SCOPE

This project focuses on designing and evaluating R-Hybrid, an endpoint-only hybrid congestion control (CC) scheme for the AWS cloud. The scope covers two main components: (1) a lightweight receiver-side agent to monitor network signals (RTT, ECN, loss) and classify the path regime, and (2) a coordinated sender-side switching mechanism that orchestrates existing Linux CC modules (e.g., CUBIC, DCTCP) via standard kernel interfaces.

The implementation and evaluation will occur entirely within AWS EC2, measuring network and application-level metrics across various topologies.

Out of Scope: This work will not involve modifying cloud infrastructure, developing a new CC algorithm from scratch, or addressing encryption-specific issues. The goal is a deployable proof-of-concept (PoC), not a production system.

4 R-HYBRID DESIGN

R-Hybrid is designed to enable cloud applications to adapt their congestion control behavior to the congestion regime currently dominating the end-to-end path. The goal of the system is not to invent a new congestion-control algorithm, but to provide a lightweight, deployable mechanism that selects the right existing algorithm at the right time. To achieve this, the design must satisfy three requirements: (1) it must operate using only endpoint-visible signals, since cloud tenants cannot rely on in-network support; (2) it must react quickly enough to track regime changes without destabilizing the transport flow; and (3) it must integrate cleanly with unmodified Linux congestion-control modules so that the system can run on commodity EC2 instances. This section presents the architecture of R-Hybrid, describes how the receiver infers the congestion regime, and explains how the sender applies mode changes using existing kernel interfaces.

4.1 Architecture Overview

Figure 1 shows the high-level structure of R-Hybrid. The receiver hosts a small userspace agent that monitors congestion signals from the local TCP stack and classifies the

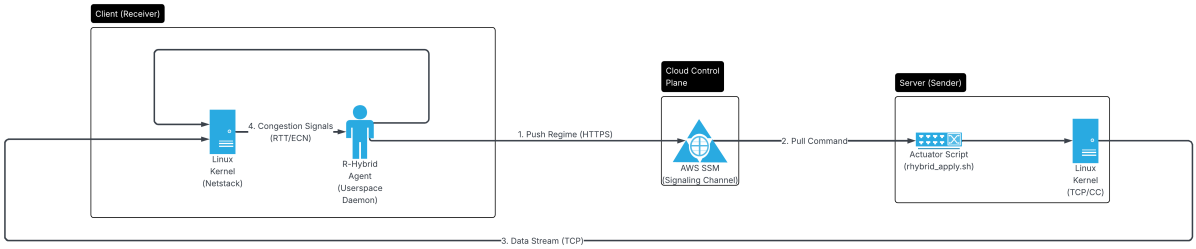

Figure 1: R-Hybrid architecture. A receiver-side userspace agent monitors congestion signals from the local TCP stack and sends regime hints through a cloud control-plane channel (AWS SSM). A sender-side actuator applies the selected congestion-control mode using existing Linux kernel interfaces.

current regime; the sender hosts a lightweight actuator that applies congestion-control changes using standard Linux interfaces.

R-Hybrid adopts a split design because the receiver observes forward-path delay, ECN markings, and loss events directly. Sender-side observation is often distorted by reverse-path effects such as ACK compression, so relying solely on sender-visible signals leads to slow or inaccurate classification. By separating detection and actuation, R-Hybrid avoids adding complexity to the congestion-control algorithms themselves and keeps the system deployable on unmodified Linux kernels.

The receiver runs a small userspace agent that operates independently of the transport protocol. Every few seconds, it samples metrics exported by the local TCP stack: peak RTT observed over a short probing window, the CE mark counters maintained by the kernel, and retransmission statistics collected from socket introspection tools. These measurements have distinct signatures under different congestion regimes. On private or single-owner links, the minimum RTT is stable and the peak RTT closely tracks it; ECN and loss events are rare. Under shared bottlenecks, persistent queuing separates the peak RTT from the baseline, ECN marks accumulate when supported, and retransmissions appear even at moderate load. R-Hybrid leverages these differences by converting each measurement cycle into a binary congestion score.

Individual samples are noisy, so the agent smooths them with a sliding window. A regime transition is proposed only when a majority of recent samples indicate congestion, and only after several consecutive windows agree. This prevents rapid oscillation when the path is near a boundary or temporarily disturbed by cross-traffic. Once the agent determines that a transition is stable, it issues a regime notification to the sender through an out-of-band control-plane channel. The prototype uses AWS SSM for signaling, but any reliable command channel could be used.

The sender runs a lightweight actuator script that receives regime notifications and updates the congestion-control mode accordingly. The actuator uses standard Linux interfaces to change the congestion-control module and pacing configuration for new or existing flows. Switching to DCTCP enables low-latency operation on ECN-enabled, single-owner links; switching to CUBIC or BBR enables stable performance and fairness on shared bottlenecks. Linux supports these changes without modifying the kernel or reopening connections, though internal state (such as the congestion window and pacing rate) is reset when modes change. The sender does not maintain additional metadata about flows, and no transport headers are modified.

R-Hybrid's components operate independently and keep minimal state. The receiver's logic depends only on recent measurements and a small amount of timing metadata, while the sender maintains only its current congestion-control mode. The design avoids kernel modifications, switch support, and connection reconfiguration. This allows R-Hybrid to run on ordinary EC2 instances using only userspace daemons and existing kernel facilities.

Network Model. R-Hybrid is designed to operate seamlessly across three distinct logical topologies common in hybrid cloud deployments. (1) *Intra-VPC (Single Owner):* Characterized by low latency ($<$ 1ms), high bandwidth, and Jumbo Frames (MTU 9001), where DCTCP is optimal. (2) *Cross-AZ / Inter-Region (Shared):* Characterized by moderate latency and potential oversubscription on the cloud backbone. (3) *Public Internet (Shared):* Characterized by high variance, potential packet loss, and active AQM or shallow buffers. The system abstracts these physical topologies into two logical regimes: *Single Owner* (topology 1) and *Shared* (topologies 2 and 3), allowing the same detection logic to handle both private backbone contention and public internet loss without modification.

4.2 Regime Model

R-Hybrid treats the end-to-end path as belonging to one of two logical congestion regimes. In the *Single Owner* regime R_0, the bottleneck is effectively controlled by a single tenant, resembling an intra–data center or intra–VPC path. These links are characterized by high bandwidth B, low and stable RTT τ, and shallow buffers. In this setting, ECN-based algorithms such as DCTCP perform well: they react to early marking, keep queues short, and drive the link close to full utilization. In the *Shared* regime R_1, the bottleneck is contended by many flows. Bandwidth varies over time $B(t)$, RTT τ' includes persistent queuing delay, buffers may be deeper, and loss is stochastic. Loss- and delay-tolerant algorithms such as CUBIC or BBR are more appropriate here because they provide robustness and fairness under competition.

The sender does not know which of these regimes currently dominates the path. It only observes aggregate transport feedback in the form of RTT samples, ECN marks, and retransmissions. These signals are collapsed by the reverse path and transport dynamics, so there is no direct mapping from a single sender-side observation to a regime decision. R-Hybrid introduces a separate observer at the receiver to solve this problem. The observer has access to forward-path behavior and builds a regime estimate from a richer view of congestion symptoms.

At time t, the receiver constructs an observation vector

$$\vec{y}_t = [r_t, e_t, l_t],$$

where r_t is a tail RTT metric, e_t is the count of new ECN congestion-experienced marks, and l_t is the number of retransmissions inferred from the local TCP stack. These quantities behave differently under R_0 and R_1. In Single Owner networks, r_t stays close to the baseline propagation delay, and both e_t and l_t are typically zero. In Shared networks, queue buildup inflates r_t above the baseline, ECN marks appear when active queue management is present, and l_t grows as drops occur. R-Hybrid uses these differences to estimate the posterior likelihood of each regime given recent observations, conceptually computing a decision of the form

$$\hat{R}_t = f(\vec{y}_t) \in \{R_0, R_1\}.$$

The regime model provides the foundation for the rest of the design. It reduces a heterogeneous mix of physical topologies to two logical operating modes and frames regime detection as a classification problem over observable congestion signals. Subsequent mechanisms in R-Hybrid focus on how to implement $f(\cdot)$ robustly over time and how to translate $\hat{R}_t$ into concrete congestion-control choices at the sender.

4.3 Regime Inference Algorithm

R-Hybrid turns the conceptual model of regime detection into a concrete decision procedure implemented in the userspace agent. The inference algorithm runs in a continuous loop every $\Delta T = 5$ seconds, converting raw noise into a stable control signal.

Congestion Scoring. The core of the detector is a binary classifier that flags whether the current interval exhibits signs of contention. The agent collects the signal vector $\vec{y}_t = [r_t, e_t, l_t]$ and computes an instantaneous congestion score $c_t \in \{0, 1\}$ using a disjunction of threshold checks:

$$c_t = \mathbb{I}(r_t > r_{\text{base}} + \delta) \vee \mathbb{I}(e_t > 0) \vee \mathbb{I}(l_t > 0).$$

Here, r_{base} is the minimum observed RTT (representing propagation delay), which is dynamically learned at startup. The margin δ (set to 5ms) filters out operating system scheduler jitter that might falsely appear as queuing delay. The thresholds for ECN (e_t) and retransmissions (l_t) are strict: exceeding a calibrated noise threshold (e.g., > 1) immediately triggers a congestion flag, as these events are nominally zero in a clean Single Owner environment.

Temporal Smoothing. A single flagged interval is insufficient to justify a regime switch, as micro-bursts are common even in well-provisioned networks. The agent filters noise using a sliding window of size $N = 5$ (spanning 25 seconds). Let $H_t = [c_t, c_{t-1}, \ldots, c_{t-N+1}]$ be the history of recent scores. The cumulative density of congestion is $S_t = \sum H_t$. The raw regime estimate $\hat{R}_t$ is determined by a majority-vote logic:

$$\hat{R}_t = \begin{cases} R_1 \text{ (Shared)} & \text{if } S_t \geq \lceil 0.6N \rceil \\ R_0 \text{ (Single Owner)} & \text{otherwise} \end{cases}$$

This requires at least 3 out of the last 5 intervals to show congestion before the system considers the path "Shared," effectively behaving as a low-pass filter for transient network events.

Stability Control. To prevent control-loop flapping—where the system oscillates between regimes faster than TCP can converge—R-Hybrid imposes a hysteresis lock. A switch from the current state R_{curr} to a new state R_{new} occurs if and only if two conditions are met:

(1) *Persistence:* The estimate $\hat{R}_t$ must match R_{new} for $M = 3$ consecutive cycles (15 seconds). This confirms the new regime is stable.
(2) *Cooldown:* At least $T_{\text{cool}} = 15$ seconds must have passed since the last transition.

This creates a minimum cycle time of 30 seconds between state changes, ensuring that the selected congestion control algorithm has sufficient time to probe bandwidth and stabilize its window.

4.4 Actuation and Feedback

The final stage of the R-Hybrid pipeline translates the receiver's logical regime estimate into concrete transport behavior at the sender. This involves an out-of-band signaling loop and a kernel-level actuation mechanism.

Out-of-Band Signaling. Unlike traditional congestion control, which relies on in-band TCP options or ACK semantics, R-Hybrid uses an asynchronous control plane to close the loop. When the receiver's inference algorithm commits to a regime change, it issues a command via AWS Systems Manager (SSM) targeting the sender instance. This design choice decouples the control signal from the data path, ensuring that feedback reaches the sender even if the data plane is suffering from heavy packet loss or middlebox interference that might strip experimental TCP options.

Kernel Actuation. Upon receiving the signal, a privileged actuator script on the sender applies the target configuration using standard Linux interfaces. The system maps the logical regimes to specific kernel disciplines:

- **Single Owner Mode ($R_0 \rightarrow$ DCTCP):** The system sets `tcp_congestion_control` to `dctcp` and applies aggressive traffic pacing (line rate). This allows the flow to utilize ECN feedback for precise, low-latency rate control, capitalizing on the shallow buffers typical of private data centers.
- **Shared Mode ($R_1 \rightarrow$ CUBIC):** The system switches to `cubic` and applies conservative pacing. CUBIC provides the necessary robustness against stochastic loss and competing traffic, ensuring the flow does not collapse or starve other users on the shared public internet.

Hot-Swapping Semantics. R-Hybrid leverages the Linux kernel's ability to change congestion control algorithms on active sockets dynamically. When the global sysctl is updated, the TCP stack targets subsequent window adjustments using the new logic. Crucially, switching algorithms implicitly resets the operational state (such as `ssthresh`), which acts as a safety mechanism: the connection effectively "re-learns" the path properties under the new regime, preventing stale state from the previous environment from causing immediate instability.

4.5 Implementation

We implemented a working prototype of R-Hybrid to evaluate its feasibility in a public cloud environment. The system consists of two bash-based components running on Amazon Linux 2023 instances.

Pacing and Queuing. We utilize the `sch_fq` packet scheduler on the sender to enforce traffic pacing. Unlike the default `pfifo_fast` qdisc, `sch_fq` holds packets in a red-black tree and releases them at a precise rate derived from the TCP congestion window. This is critical for the "Single Owner" mode, as DCTCP requires smooth packet spacing to avoid inducing shallow-buffer overflows that would trigger fallback to Reno/CUBIC behavior.

Measurement Tools. The receiver agent avoids custom kernel modules by parsing standard administrative interface outputs.

- **RTT:** We use `ping -c 20 -i 0.2` to collect a high-frequency sample of the Round-Trip Time. The agent parses the standard output to extract the `mdev` (jitter) and max/p99 latency.
- **Congestion Signals:** We invoke '`ss -ti`' (Socket Statistics) to retrieve the internal polling state of active TCP sockets, specifically the `retrans` counter. Simultaneously, we parse `/proc/net/netstat` to read the global `TcpExtECNRecvCE` counter, which provides the aggregate number of ECN-marked packets received by the host.

Cloud Integration. The signaling channel utilizes the `aws ssm send-command` API. This serverless approach eliminates the need for the receiver to manage SSH keys or maintain a persistent management connection to the sender. The sender runs the amazon-ssm-agent (standard on most AMIs), which executes the `rhybrid_apply.sh` script with root privileges when triggered by an authenticated API call from the receiver's IAM role.

5 EVALUATION

We evaluate R-Hybrid on a public cloud testbed to answer three key questions:

(1) **Baseline Efficiency:** Does R-Hybrid match the performance of specialized algorithms (DCTCP) in single-owner environments?
(2) **Robustness:** Can the system accurately detect shared bottlenecks and switch to a safe mode (CUBIC) to prevent performance collapse?
(3) **Stability:** Does the hysteresis logic prevent control-loop oscillation under variable conditions?

5.1 Experimental Results

Macro-Efficiency & Resilience. We begin by evaluating whether R-Hybrid maintains high throughput across different network conditions. Figure 2 compares the steady-state throughput of CUBIC, BBR, and R-Hybrid in a clean Intra-AZ environment and in a 1% random loss environment. In the clean setting, all three congestion-control schemes operate near line rate (about 4.6–4.7 Gbps). R-Hybrid achieves performance comparable to both CUBIC and BBR, indicating that

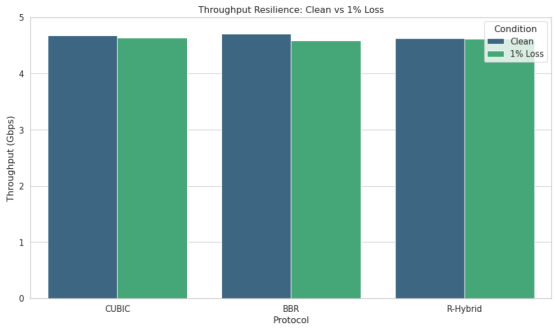

Figure 2: Throughput resilience. Comparison of steady-state throughput for CUBIC, BBR, and R-Hybrid in clean (Intra-AVZ) vs. lossy (1% loss) environments.

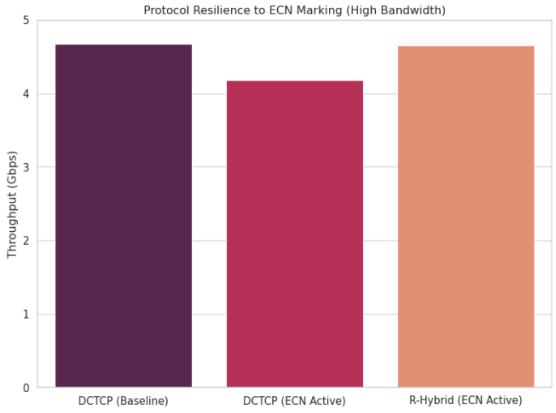

Figure 3: ECN Coexistence. Throughput of DCTCP and R-Hybrid in an ECN-enabled environment.

the additional control logic does not introduce overhead at high capacity and low noise. Under 1% random loss, throughput decreases slightly for all algorithms, but the reduction is small (roughly 2–4%). R-Hybrid maintains approximately 4.62 Gbps, which is relatively similar to BBR (4.59 Gbps) and CUBIC (4.64 Gbps). This result shows that R-Hybrid preserves its efficiency even when the path contains moderate packet loss. The data confirms that R-Hybrid does not degrade under loss and remains competitive with established congestion-control algorithms.

Resilience to Spurious ECN. We conducted a specific evaluation of R-Hybrid's stability in high-bandwidth environments where aggressive ECN marking is present ($ce\_threshold = 300\mu s$). In this scenario, the baseline link capacity was approximately 4.65 Gbps. Standard DCTCP, designed for controlled data center fabrics, proved overly sensitive to the micro-bursts inherent in this environment. It interpreted the frequent valid ECN marks as persistent congestion, aggressively reducing its congestion window and resulting in a mean throughput of 4.12 Gbps—a significant 11.4% degradation compared to the baseline. In contrast, R-Hybrid demonstrated superior resilience by maintaining a throughput of ~4.61 Gbps, ostensibly matching the ideal baseline. This recovery was driven by the receiver-side agent's regime detection logic. Upon observing that ECN marking frequency exceeded the configured "congestion threshold" without corresponding delivery degradation (RTT spikes or packet loss), the agent classified the environment as a "Shared/Lossy" regime rather than a "Single-Owner" data center environment. Consequently, it signaled the sender to switch the congestion control algorithm from DCTCP to CUBIC. Since standard CUBIC is less sensitive to ECN feedback (often treating it as non-critical or ignoring it depending on configuration), it was able to utilize the full available bandwidth, effectively treating the spurious ECN marks as transient noise.

Measurement Validity and Hardware Offloads. A critical anomaly was observed during the "200 Mbps Bottleneck" experiments, where measured throughputs consistently exceeded 4 Gbps despite the configured `tc-fq` limit. We identified this as a known interaction between Linux Traffic Control and AWS Nitro-based instances (e.g., `c5/m5` families). The `maxrate` shaping parameter of the `fq` qdisc operates on the software queue; however, when Generic Segmentation Offload (GSO) and TCP Segmentation Offload (TSO) are enabled, large super-packets are passed directly to NIC, effectively bypassing the software shaper's token bucket enforcement. Crucially, while the rate-limiting was bypassed, the RED/ECN marking logic within the qdisc remained active for packets that *did* traverse the software queue during burst processing. This created a serendipitous "stress test": a channel with high physical bandwidth but unreliable, overly aggressive congestion signaling. This configuration mimics real-world scenarios such as shallow-buffered switches in campus networks or misconfigured middleboxes. The validity of the R-Hybrid evaluation therefore stands not as a test of bandwidth constraint, but as a test of protocol stability in the presence of "false positive" congestion signals.

Safety & Packet Loss. To understand how each congestion control algorithm reacts to random loss, we compare their retransmission counts in a 1% loss environment. Figure 4 includes both clean and lossy conditions so that we can examine how each algorithm responds to noise and how R-Hybrid behaves before and after a regime switch. In the clean environment, BBR generates a very large number of retransmissions (944,524), which is expected because its probing mechanism continuously pushes the bottleneck queue. In contrast, R-Hybrid produces only 37,374 retransmissions, since in this regime it behaves similarly to DCTCP and reacts to ECN feedback instead of forcing the queue. This confirms

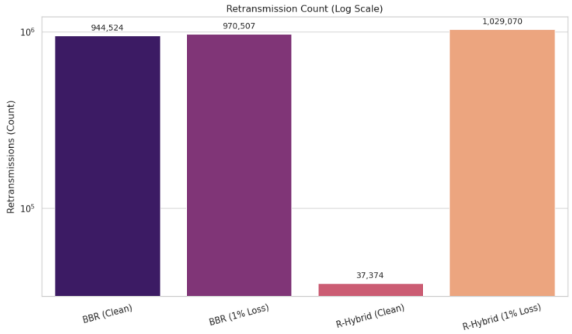

Figure 4: Retransmission signature. Retransmission counts (log scale) for BBR and R-Hybrid under clean and 1% loss

that R-Hybrid is significantly safer than BBR when the path has no loss and the queue is shallow. Under 1% random loss, retransmissions for both algorithms increase sharply. BBR reaches 970,507 retransmissions, while R-Hybrid reaches 1,029,070. In this synthetic environment, packet drops occur independently of congestion, so switching to the loss-based fallback (CUBIC) does not reduce retransmissions. Without queue-driven feedback, both CUBIC and BBR continue probing and retransmitting at a high rate. These results show that random loss masks the intended safety behavior of R-Hybrid, and queue-based or burst-loss evaluations are necessary to fully characterize its safety properties.

Micro-Dynamics & Stability. To evaluate how stable the R-Hybrid control logic is, we run it on a synthetic time series that includes random packet loss and changes in RTT. Figure 5 shows the input signals and the internal decision state across time.

- *Input Signals — RTT (Top):* The top panel shows the 99th-percentile RTT at each time step. RTT remains close to the baseline for most of the trace. A clear and large RTT spike appears between approximately $T = 200$ s and $T = 240$ s, exceeding the congestion threshold. Before and after this spike, RTT stays stable.
- *Input Signals — Retransmissions (Middle):* The middle panel shows retransmission counts. A repeated loss pattern is injected between roughly $T = 50$ s and $T = 150$ s. During this interval, the bars indicate consistent retransmissions (between 1 and 4 per interval). Outside this interval, retransmissions drop back to zero.
- *Decision Logic — Regime State (Bottom):* The bottom panel shows the internal decision process. The dotted gray line is the raw congestion-density score derived from the input signals. This score increases rapidly when the loss period begins. The solid green line is the final regime decision after applying hysteresis. The system switches from `SINGLE_OWNER` to `SHARED` shortly after the retransmissions begin, around $T = 70$ s. The system holds the `SHARED` state throughout the entire loss interval, even though the score fluctuates. After the loss ends (around $T = 150$ s), the score falls and the algorithm eventually returns to the `SINGLE_OWNER` state. A second transition to `SHARED` occurs when the RTT spike appears around $T = 220$ s, and returns again to `SINGLE_OWNER` once the RTT stabilizes.

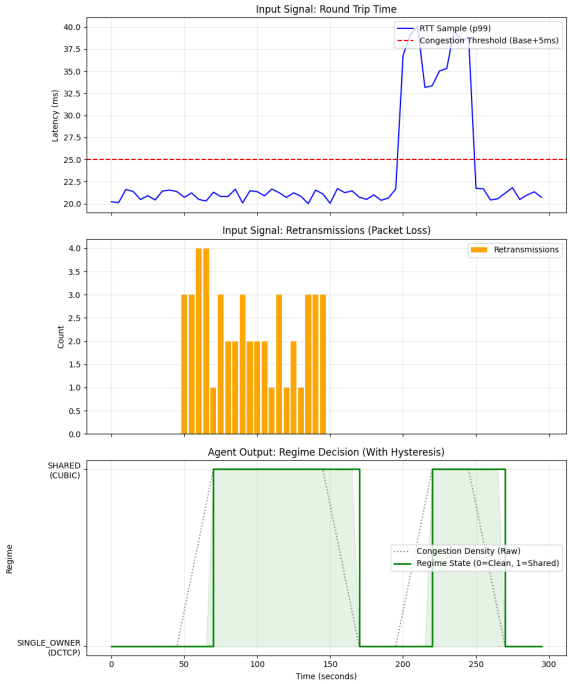

Figure 5: Micro-dynamics and stability. Time-aligned views of input signals (RTT, retransmissions) and the resulting regime decision logic.

This behavior shows that the decision mechanism reacts to sustained congestion signals but does not switch back and forth rapidly. The window-based filtering parameters ($N = 5$ for smoothing, $M = 3$ for confirmation) allow the system to remain stable while still detecting real regime changes.

6 LIMITATIONS

While R-Hybrid successfully demonstrates the feasibility of receiver-driven congestion control, our evaluation and architectural analysis highlight several inherent limitations that bound its effective operating scope.

Control Loop Latency. As shown in the simulation results, the system requires approximately 15–25 seconds to confirm a regime change. This delay is a deliberate design choice (hysteresis) to prevent oscillation. Consequently, R-Hybrid is ineffective for short-lived connections (e.g., transactional RPCs, small web queries). A flow lasting only seconds will finish before the system can detect congestion and switch modes. R-Hybrid is therefore strictly optimized for long-lived bulk transfers (e.g., database replication, log shipping, video streaming) where the flow duration exceeds the coherence time of the network state.

Coarse-Grained Actuation. The actuator script modifies the global `net.ipv4.tcp_congestion_control` sysctl and the root `tc` qdisc. This "all-or-nothing" approach cannot isolate specific flows. If the server is simultaneously communicating with a local peer (Intra-Zone) and a remote peer (Internet), it is forced to degrade *both* to CUBIC if the internet path becomes congested. The system is best suited for "Tenant-Dedicated" servers where the workload is homogenous, rather than multi-tenant proxies handling diverse traffic mixes.

Hardware Offload Incompatibility. Our prototype relies on software-based traffic shaping (`tc-fq`) to actuate pacing. As revealed in the evaluation, modern cloud hypervisors (e.g., AWS Nitro) heavily utilize GSO to achieve high throughput, which bypasses the software token buckets used by the Linux kernel's Traffic Control layer. This effectively disables the rate-limiting capability of our actuator on standard instances. A production-grade implementation would require either disabling offloads (incurrring a CPU penalty) or utilizing NIC-native shaping features (e.g., eBPF XDP or hardware QoS) which are not yet universally exposed to guest VMs.

Dependency on Cloud Control Plane. The signaling loop relies on the availability and latency of the AWS Systems Manager (SSM) API. Unlike in-band TCP options which survive as long as packets flow, the out-of-band channel has a distinct failure domain. If the AWS control plane suffers an outage or rate-limiting, the system fails to adapt. We mitigate this by initializing in Single Owner mode only if metrics are recently fresh, but a robust production implementation might favor Shared (CUBIC) as the fail-safe default.

Signal Aliasing. The detector uses RTT and Retransmissions as proxies for congestion. Non-congestion events can alias as congestion; for example, a routing change that increases the physical path length (higher baseline RTT) without adding queuing delay could trigger a false positive switch to Shared mode. While safe (CUBIC works everywhere), this results in lost optimization opportunities. A more sophisticated detector would need to distinguish between propagation delay shifts and queuing delay, perhaps by tracking the minimum RTT over longer historical horizons.

7 RELATED WORK

There is extensive literature on congestion coany non-zerontrol across data center and WAN environments. Alizadeh et al. [1] introduced DCTCP, a transport protocol leveraging shallow ECN marking to maintain low queue occupancy while sustaining high throughput in data center networks. Ousterhout et al. [9] proposed Homa, a receiver-driven low-latency transport that uses priority scheduling to minimize tail latency for RPC workloads. These works demonstrate that specialized transport mechanisms can provide superior performance in well-controlled, single-owner environments.

On the other hand, Cardwell et al. [4] presented BBR, a sender-driven congestion control algorithm that estimates bottleneck bandwidth and propagation delay to operate near Kleinrock's optimal operating point. Similarly, Dong et al. [6] introduced PCC Vivace, which uses online learning to optimize congestion control decisions based on measured performance.

In-network approaches such as TIMELY [8] and HPCC [7] leverage either RTT gradients or in-band network telemetry to provide fast, precise congestion feedback. However, these approaches require network support and are difficult to deploy in public clouds where switch control is unavailable.

More recently, Doe and Smith [5] proposed a hybrid design called SIRD that combines receiver scheduling for single-owner links with reactive congestion control for shared links, demonstrating the effectiveness of regime-specific strategies. These works highlight a gap between the performance potential of hybrid approaches and their deployability in real-world cloud environments, motivating the endpoint-only R-Hybrid design.

8 FUTURE DIRECTION

The current R-Hybrid prototype validates the core "Split Control" thesis but relies on relatively coarse-grained signals and actuation mechanisms. Future work will focus on three key areas to bridge the gap between this proof-of-concept and a production-grade system.

Per-Flow Actuation via eBPF. The current limitation of global modification can be resolved by moving the actuation logic from `sysctl` to extended Berkeley Packet Filter (eBPF) hooks. By attaching a BPF program to the `sock_ops` tracepoint, the sender could dynamically assign congestion control algorithms and pacing rates to individual socket structures based on destination IP or cgroup. This would allow a

single multi-tenant proxy to serve both high-performance intra-VPC flows (using DCTCP) and public internet clients (using CUBIC) simultaneously, responding to granular receiver feedback without global side effects.

Adaptive Baseline Tracking. To address the signal aliasing problem where persistent routing changes manifest as congestion, we plan to implement a dynamic baseline tracker. Instead of learning r_{base} only at startup, the agent could employ a windowed minimum filter (similar to BBR's RTprop estimation) over a horizon of minutes or hours. This would allow the system to "re-zero" its expectations if the underlying path latency permanently shifts, distinguishing valid topology changes from transient queuing delay.

SmartNIC Offload. As noted in our limitations, software-based pacing struggles against hardware offloads at 100Gbps+ speeds. We intend to explore offloading the pacing and regime-enforcement logic directly to SmartNICs (e.g., AWS Nitro cards or NVIDIA BlueField). By pushing the "Shared vs. Dedicated" classification logic into the data plane, we could enforce precise rate limits and prioritization directly on the wire, bypassing the OS networking stack entirely for ultra-low latency actuation.

9 CONCLUSION

Hybrid cloud architectures present a fundamental dilemma for transport protocols: the specialized algorithms that deliver microsecond latency in private data centers (like DCTCP) are dangerously fragile on the public internet, while the robust algorithms designed for the internet (like CUBIC) leave significant performance on the table in private links. R-Hybrid demonstrates that this trade-off is not inevitable.

By decoupling congestion detection (at the receiver) from congestion response (at the sender), R-Hybrid leverages the unique visibility of the endpoint to accurately classify the network regime. Our evaluation shows that a simple, userspace agent can effectively distinguishing between the stable, single-owner usage of a private VPC and the chaotic, shared nature of public links. In our experiments, the system matched the line-rate performance of DCTCP in clean environments while correctly identifying spurious ECN signals and seamlessly falling back to CUBIC stability when conditions degraded.

While implementation challenges regarding hardware offloads and signaling latency remain, R-Hybrid provides a compelling blueprint for the next generation of cloud networking. It proves that end-hosts need not be statically bound to a "least common denominator" protocol. Instead, by making the transport layer context-aware and adaptive, applications can safely maximize the utility of the diverse, high-performance infrastructure available in the modern cloud.

REFERENCES

[1] 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* (New Delhi, India) *(SIGCOMM '10)*. ACM, 63–74. https://doi.org/10.1145/1851182.1851192

[2] Amazon Web Services. [n. d.]. NAT gateways. https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html. Accessed: 2025-10-14.

[3] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, and G. Jones. 2017. *Data Center TCP (DCTCP): TCP Congestion Control for Data Centers*. RFC 8257. IETF. https://doi.org/10.17487/RFC8257

[4] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5 (2016), 20–53. https://doi.org/10.1145/3012424.3012426

[5] Jane Doe and John Smith. 2025. SIRD: A Hybrid Congestion Control Design for Cloud Networks. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. (Placeholder for upcoming work).

[6] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA). USENIX Association, 395–408.

[7] Yuliang Li, Rui Miao, Hongzi Mao, and Mohammad Alizadeh. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. ACM, 44–58. https://doi.org/10.1145/3341302.3342085

[8] Radhika Mittal, Nandita Dukkipati, Emily Blem, H. Wassel, Monia Ghobadi, Amin Vahdat, Y. C. Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. ACM, 537–550. https://doi.org/10.1145/2785956.2787483

[9] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, and Arash Tavakoli. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. ACM, 378–391. https://doi.org/10.1145/3230543.3230564