A Case for CATS: A Conductor-driven Asymmetric Transport Scheme for Semantic Prioritization

Syed Muhammad Aqdas Rizvi – Lahore University of Management Sciences (LUMS)
25100166@lums.edu.pk

August 23, 2025

Abstract

The demand for improved Quality of Experience in interactive applications, whether for web browsing or Extended Reality and remote control, necessitates a more intelligent approach to data delivery than traditional FIFO transport. This proposal introduces CATS (Conductor-driven Asymmetric Transport Scheme), a framework for modifying TCP to be aware of the semantic importance of the data it carries. By implementing a priority-aware scheduling mechanism directly within the transport layer, CATS aims to dramatically improve end-user perceived performance by delivering critical content first.

This research directly addresses known limitations in the protocol stack, such as the high overhead of parallel TCP connections in HTTP/1.1 and the transport-layer head-of-line blocking that impacts HTTP/2. The proposed mechanism, which is designed to be deeply integrated with a modern congestion control algorithm like BBR, also presents a transport-native design philosophy for prioritization, exploring an alternative architectural trade-off to the application-managed stream model in QUIC. The core hypothesis is that this tightly-coupled, end-host approach can lead to more efficient and responsive scheduling by leveraging the transport layer's real-time network state awareness. This document details the architectural design of CATS, outlines a rigorous simulation-based methodology for evaluating its performance and fairness, and explores its broad applicability across civilian, defense, and future 5G/6G use cases.

1 Introduction: Description of Core Concepts

1.1 Introducing CATS: A New Framework for Transport Scheduling

The correlation between web performance and user engagement is unequivocally established [18]. However, conventional transport protocols like TCP operate on a First-In, First-Out (FIFO) basis, failing to differentiate between content elements based on their contribution to the user's perceived experience, especially under network duress. This leads to inefficient resource utilization where large, non-critical assets can delay the delivery of small, essential data.

To address this fundamental limitation, this proposal introduces **CATS (Conductor-driven Asymmetric Transport Scheme)**, a novel framework for modifying TCP's sending logic. The "Conductor-driven" term refers to the new orchestrating logic introduced into the transport layer, responsible for managing the transmission schedule. The core principle is "Asymmetric" treatment: unlike a standard FIFO queue, CATS handles data based on its application-defined semantic importance. This is implemented as a new "Transport Scheme", being a complete system of priority queuing, fairness rules, and scheduling logic.

The goal of the CATS framework is to dramatically improve the Quality of Experience for a wide range of interactive applications, such as web browsing; Extended Reality; remote desktop sessions; and tactical communications, by delivering critical content first. It also aims to enhance overall network efficiency. By providing fine-grained prioritization within a single, cooperative transport connection, CATS addresses fundamental limitations in existing protocols, such as the high overhead of parallel connections in HTTP/1.1 and the transport-layer head-of-line blocking that impacts HTTP/2. Furthermore, CATS presents a transport-native design philosophy for prioritization that contrasts with the application-managed stream model in QUIC, creating a tightly-coupled system where the transport layer's real-time network awareness can be directly leveraged for scheduling decisions.

1.2 The Strategy: Semantic Content Prioritization

The CATS framework is built upon a strategy of **Semantic Content Prioritization**. To illustrate, consider a standard news article page accessed over a slow mobile connection: the immediate value lies in the headline and article text, yet network resources might be equally consumed transmitting bytes for large banner images, delaying the primary content.

By assigning explicit priorities derived from developer annotations, server-side heuristics, or potentially automated analysis, to distinct content elements (HTML structure, critical CSS, primary text, interactive scripts, etc.), the delivery process can be intelligently orchestrated. The objectives of this strategy are to:

- **Optimize Perceived Performance:** Dramatically reduce the time required for the user to access and interact with the most critical content elements, enhancing subjective speed.
- **Enable Robust Progressive Enhancement:** Facilitate the rapid rendering of a functional baseline page, with subsequent loading of non-essential enhancements contingent on available bandwidth.
- **Facilitate Adaptive Quality Delivery:** Allow for initial delivery of lower-fidelity content (e.g., low-resolution images) with subsequent "upgrades" to higher fidelity versions if network conditions permit.
- **Support Constrained Data Scenarios:** Provide mechanisms to limit delivery to only the highest-priority content tiers, respecting user data caps or explicit low-data preferences.

This strategy fundamentally shifts the focus from undifferentiated byte delivery to a value-driven transmission schedule aligned with user perception.

1.3 The Venue: TCP Implementation

The proposed implementation venue for the CATS framework is directly within the **Transmission Control Protocol (TCP)**. The rationale for exploring this transport-native approach stems from the potential to leverage TCP's unique characteristics:

- **Comprehensive Network State Awareness:** TCP, particularly when employing advanced congestion control algorithms like BBR, maintains a real-time model of network path characteristics, including estimates of bottleneck bandwidth (BtlBw) and round-trip propagation time (RTprop) [13]. Integrating prioritization logic at this layer could enable feedback loops that react more rapidly and accurately to dynamic network conditions than application-level logic relying on potentially delayed or abstracted signals.

- **Direct Control Over Transmission Mechanics:** TCP manages data segmentation, send buffering, transmission scheduling (pacing), and retransmission logic [5, 38]. Implementing prioritization here offers the potential for fine-grained control over precisely which data segment is transmitted next when capacity is limited.

- **Standardization Potential:** Although achieving standardization is a significant hurdle, a successful TCP-level mechanism could offer broader applicability across diverse applications compared to potentially fragmented solutions tied to specific application protocols (e.g., HTTP variants).

This proposal seeks to rigorously assess whether these theoretical advantages manifest as practical benefits sufficient to justify the inherent challenges of modifying a foundational internet protocol. Furthermore, while strongly motivated by enhancing web performance, such a transport-level prioritization mechanism could potentially benefit a wider range of interactive TCP-based applications where data streams inherently contain elements of varying semantic importance for responsiveness.

1.4 Addressing Foundational Objections: Precedent and Philosophy

Modifying TCP to incorporate application-level semantics (content priority) immediately confronts valid architectural and practical objections, which must be addressed proactively:

- **Protocol Layering Principles:** The canonical OSI model emphasizes separation of concerns [47]. Introducing application-specific hints into the transport layer appears to violate this principle.
  Response: While layering is a vital design guideline, performance optimization in real-world systems often necessitates pragmatic cross-layer interactions. TCP's congestion control is already influenced by application sending patterns. QUIC integrates transport and encryption [29]. BBR actively models end-to-end path properties [13]. We argue that passing priority *metadata* for TCP to use in its existing resource management functions (scheduling, buffer management) represents a justifiable, pragmatic evolution, not a fundamental architectural breach, provided substantial benefits are demonstrated.

- **TCP Complexity and Ossification:** There exists a strong and valid imperative to maintain TCP's stability and resist unnecessary complexity, often termed "TCP ossification" [cf. 24].
  Response: TCP has demonstrably *not* remained static. The development and widespread deployment of complex algorithms like CUBIC [22] and particularly BBR [13] underscore the ecosystem's willingness to adopt significant internal complexity when driven by clear performance imperatives in evolving network environments [cf. 6, 16]. BBR's sophisticated path modeling logic is a testament to TCP's capacity for evolution. This proposal asserts that semantic priority awareness could be the next such performance-driven evolution, following established precedent. Further, research into synthesizing existing transport strategies for optimal performance in demanding environments like data centers, such as PASE [33], also demonstrates the continued effort to evolve and combine transport mechanisms beyond their original singular designs, supporting the idea that transport protocols can and do adapt to new requirements.

- **The Rise of QUIC/HTTP/3:** QUIC was explicitly designed to facilitate transport innovation by operating over UDP and integrating features like stream multiplexing without head-of-line blocking [29], enabling effective application-level prioritization in HTTP/3 [11, 32]. Does this render TCP modification moot?
  Response: The success of QUIC/HTTP/3 validates the *need* for robust prioritization. However,

it doesn't automatically preclude potential advantages of a TCP-integrated approach, specifically regarding the immediacy of network state feedback within the TCP stack itself. Furthermore, QUIC currently lacks standardized transport-level prioritization mechanisms applicable beyond HTTP [cf. 27]. Investigating TCP remains relevant for the vast existing TCP ecosystem and for potentially informing future QUIC evolution.

This research proceeds with full awareness of these objections, framing the investigation as an exploration of potential performance trade-offs rather than a premature assertion of TCP's superiority.

1.5 Research Objectives

The primary objectives of this proposed research are:

1. To design and specify concrete mechanisms for embedding semantic priority metadata within TCP segments and modifying TCP's transmission logic (particularly within a BBR context) to act upon this metadata while incorporating explicit fairness controls.

2. To quantitatively evaluate, through simulation and prototype implementation, the impact of the proposed mechanism on key web performance metrics (e.g., FCP, LCP [21], TTI) under a representative range of network conditions.

3. To rigorously assess the fairness implications of the proposed modifications, both within a single prioritized flow (intra-flow fairness) and between prioritized flows and competing standard TCP/QUIC flows (inter-flow fairness) using established metrics [e.g., 31].

4. To conduct a comparative performance and complexity analysis against state-of-the-art application-layer prioritization techniques, specifically HTTP/3 prioritization over QUIC.

5. To identify and analyze the practical challenges related to implementation, standardization, and deployment.

2 Technical Background

2.1 TCP Fundamentals

The Transmission Control Protocol provides the bedrock for reliable internet communication, offering connection-oriented, ordered, and error-checked byte stream delivery [38]. Its core functions include the three-way handshake, sequence numbering, acknowledgments, checksums for integrity, sliding window flow control, and sophisticated congestion control mechanisms [5] designed to prevent network collapse [30]. Critically for this proposal, standard TCP operates without intrinsic knowledge of the semantic meaning or relative importance of the application data it transports.

2.2 Evolution of TCP Congestion Control

TCP congestion control has evolved significantly. Early algorithms (Tahoe, Reno) were primarily loss-reactive. CUBIC [22] improved performance in high bandwidth-delay product networks but largely retained the loss-based reaction model. TCP BBR [13] marked a fundamental shift to a model-based approach, actively probing the network to estimate BtlBw and RTprop. BBR attempts to control the amount of data in flight to match the estimated BDP, cycling through distinct phases (STARTUP, DRAIN, PROBE_BW, PROBE_RTT) to manage throughput and queuing delay [12]. This inherent network awareness makes BBR a particularly interesting foundation for integrating priority-based decisions.

2.3 Application-Layer Prioritization Solutions

Recognizing the need for prioritization, application protocols have incorporated relevant mechanisms:

- **HTTP/2** introduced stream multiplexing, header compression (HPACK), server push, and stream prioritization based on dependencies and weights [10]. However, its effectiveness is often hampered by inconsistent implementations and, fundamentally, by TCP's head-of-line blocking phenomenon, where the loss of a single TCP segment can stall all multiplexed HTTP/2 streams.
- **HTTP/3 and QUIC** represent a significant advance. By running over QUIC [29], which provides multiple independent, individually flow-controlled transport streams, HTTP/3 [11] largely overcomes the transport-level HoL blocking issue. This allows the application-level prioritization schemes (e.g., as defined in RFC 9218 [32]) to operate much more effectively [16]. QUIC also introduces benefits like reduced connection latency and integrated encryption.

2.4 Limitations of Existing Network-Layer QoS

IP-level mechanisms like the Differentiated Services Code Point (DSCP) field [34] allow packets to be marked for preferential treatment. However, their efficacy for end-to-end application prioritization across the public internet is limited due to inconsistent honoring of these markings by intermediate routers and the fundamental disconnect between network-layer packet handling and transport-layer stream semantics.

2.5 Related Work in End-Host and In-Network Scheduling

While our primary focus is on enhancing user-perceived performance for general internet traffic, the principles of prioritization and intelligent queue management are fundamental to network performance research. This work is informed by and situated within two major areas of research: end-host transport scheduling in datacenters and in-network Active Queue Management (AQM).

Datacenter Transport Scheduling

The need for transport-level scheduling is most evident in datacenter networks, where optimizing for metrics like Flow Completion Time (FCT) is necessary. Early work like PDQ established the benefits of priority and deadline-aware queuing for latency-sensitive flows [25]. This was followed by influential schemes like pFabric [4] and PIAS [7], which demonstrated significant performance gains by incorporating concepts such as flow size awareness, preemption, and information-agnostic scheduling. Furthermore, work on PASE [33] explores synthesizing existing transport strategies for near-optimal performance. While these systems are tailored for the unique characteristics of datacenter traffic (e.g., many short flows, low RTTs), they establish a strong precedent for the effectiveness of modifying transport behavior to achieve specific performance objectives through prioritization. The work herein draws inspiration from this broader willingness to evolve transport mechanisms while targeting the different problem domain of semantic importance in user-facing applications over general internet paths.

Active Queue Management and Bufferbloat

A parallel line of research focuses on intelligent queue management within network routers to combat bufferbloat and improve fairness. Active Queue Management (AQM) schemes are designed to keep queuing delays low and manage network congestion proactively.

- **CoDel (Controlled Delay)** is an AQM that drops packets based on their sojourn time in a queue, effectively preventing buffers from growing excessively long [35].
- **CAKE (Common Applications Kept Enhanced)** is a an AQM that combines CoDel with fairness queuing and other mechanisms to provide a comprehensive traffic management solution, particularly for consumer-grade links [26].

This body of work is highly relevant as it addresses the problem of queuing-induced latency from an *in-network* perspective. The CATS **Conductor**, the orchestrating logic at the end-host, can therefore be viewed as a source-based complement to in-network AQM. While CoDel and CAKE manage congestion at the bottleneck router, CATS ensures that the most semantically important data is sent from the source first, preventing it from even contributing to unnecessary queuing at downstream routers.

Fairness and Protocol Coexistence

A critical challenge for any new transport mechanism is ensuring its fair coexistence with existing protocols. Research in this area has explored how to manage shared network resources to prevent one protocol from unfairly dominating others. For instance, modified AQM schemes like a modified CHOKe algorithm have been proposed to enable the fair coexistence of different transport protocols, such as TCP and DCTCP, within a single datacenter environment [28]. This work emphasizes the importance of the inter-flow fairness analysis in our research methodology, which must rigorously evaluate CATS' impact on the broader network ecosystem.

3 High-Level Design and Architecture

3.1 Priority Definition and Signaling (Application Domain)

The assignment of priorities occurs before data reaches TCP:

- **Assignment:** Priorities (e.g., numerical levels: 0 = Critical, 1 = High, 2 = Normal, 3 = Low, 4 = Background) are assigned by developers (via HTML attributes like fetchpriority [44] or custom data-transport-priority, CSS comments, JavaScript framework integration), server-side logic (based on content type, URL patterns, heuristics), or potentially advanced models (ML/LLM analysis, with performance considerations).

- **API Extension:** A mechanism is required to pass this priority metadata from the application to the TCP stack alongside the data buffers submitted via the socket API (e.g., extending sendmsg() with control messages or defining a new setsockopt() level).

Developer Mechanisms for Priority Assignment

To make the priority assignment concrete, consider these illustrative mechanisms a developer might use:

- **HTML Attributes:** Leveraging existing or custom attributes.
  - Using the standard fetchpriority hint [44] primarily informs the browser's fetch scheduler, but a server could potentially use this hint to infer transport priority:

    ```
    <link rel="stylesheet" href="critical.css"
        fetchpriority="high">
    <img src="hero.jpg" fetchpriority="high">
    <script src="analytics.js"
        fetchpriority="low"></script>
    ```

  - Using custom `data-` attributes to explicitly signal desired transport priority, potentially alongside adaptive loading hints:

    ```
    <img src="high-res.jpg" data-transport-priority="low"
         data-lowres-src="low-res.jpg"
         data-transport-priority-lowres="medium">
    ```

    In this hypothetical example, the server application would parse these attributes and associate the corresponding priority level when sending image data chunks to the TCP socket.

- **Conceptual JavaScript Framework Integration:** Modern frameworks could abstract priority assignment.
  - Components might accept priority props:

    ```
    <Image src="background.png"
        transportPriority={PRIORITY.BACKGROUND} />
    <ArticleText content={...}
        transportPriority={PRIORITY.CRITICAL} />
    ```

  - Data fetching libraries could allow specifying priority:

    ```
    fetchResource('/api/data',
        { transportPriority: PRIORITY.HIGH });
    ```

  These framework-level assignments would translate into setting the appropriate priority via the extended socket API when the underlying data is transmitted.

- **Server-Side Logic:** A backend application or CDN edge worker could determine priority based on request properties or content analysis before writing data to the socket:

```
if (isCriticalApiPath(request.url)) {
  socket.write(data, { transportPriority: PRIORITY.CRITICAL });
} else {
  socket.write(data, { transportPriority: PRIORITY.NORMAL });
}
```

It is important to distinguish this proposed mechanism from purely application-layer HTTP prioritization schemes like RFC 9218 [32]. While HTTP headers defined in such schemes could *inform* the server application's decision on what priority to signal to the TCP layer via the socket API, the headers themselves are not directly interpreted by TCP. The priority information must be associated with the data buffers passed to the transport layer.

Additionally, this concept of decomposing a larger application-level task (e.g., rendering a webpage) into components of varying importance aligns conceptually with research in other domains, such as coflow scheduling in data centers [14, 15, 19]. In coflow scheduling, multiple related network flows contributing to a single distributed computation are managed collectively to optimize overall job completion. Similarly, a webpage can be viewed as a "coflow" of resources (HTML, CSS, scripts, images), where the timely delivery of critical components dictates the user's perceived performance. This proposal aims to enable finer-grained, segment-level enforcement of such priorities within a single TCP connection carrying these diverse resources.

3.2 Embedding Priority Metadata in TCP Segments

- **Primary Approach: TCP Option.** Define a new TCP Option Kind (requiring IANA assignment). This option would be included in data-carrying TCP segments.
  Format: Must be compact due to the 40-byte total limit for all options [38]. A 2-byte option could suffice: 1 byte for Kind, 1 byte for Length=2, leaving payload bits within the Kind/Length bytes or requiring a subsequent byte for the priority value (e.g., encoding 3-5 priority levels in a few bits).
  Challenges: Option space scarcity; potential for middlebox interference (filtering or stripping unknown options) [24].

- **Alternative Approaches (Less Favored):** Signaling priority during the SYN handshake limits granularity; establishing a dedicated control channel using options adds significant complexity.

The per-segment TCP Option approach offers the most promising balance of granularity and feasibility, despite the known challenges with TCP options.

3.3 Modifying TCP Transmission Logic (Priority Enforcement within BBR Context)

The core innovation lies in modifying TCP's behavior, hypothesized here within a BBR framework:

- **Priority-Aware Send Buffer Management within the Kernel:** The core modification resides within the TCP stack, typically operating within the operating system kernel. When an application writes data to a socket using the extended API (which includes priority metadata), TCP places this data (or references to it along with its associated priority) into its kernel-managed **TCP send**

buffer. The proposed priority-aware TCP logic then implements a priority queuing discipline on this kernel buffer. Instead of strict FIFO selection, when the congestion window (cwnd), pacing rate, and receiver window (rwnd) permit transmission, the modified TCP logic preferentially selects data chunks associated with the highest available priority level from the send buffer for segmentation and subsequent transmission. Lower-priority data is only segmented and sent when no higher-priority data is pending in the buffer or as dictated by fairness mechanisms (see below). This ensures scheduling decisions happen as close to the transmission time as possible, informed by both application priority and current network conditions.

- **Integration with BBR Dynamics:**
  Bandwidth Allocation: When BBR estimates BtlBw, the priority scheduler ensures this capacity is preferentially used for segments carrying higher-priority data.
  Congestion Response Modification: Could BBR's reaction to congestion signals (loss, ECN, RTT increase suggesting queue growth) be modulated by the priority of data being sent or ACKed? For example, could the multiplicative decrease factor be slightly less severe if only high-priority data was in flight? This requires rigorous analysis to avoid unfairness.
  PROBE_BW Phase: When BBR probes for additional bandwidth, could it preferentially use higher-priority segments for these probes, ensuring critical data benefits first from newly discovered capacity?
  PROBE_RTT Phase: Ensure that the brief rate reduction during PROBE_RTT doesn't unduly delay critical, high-priority segments waiting in the buffer.

- **Facilitating Adaptive Loading:** This mechanism inherently supports adaptive strategies. The application sends data for a low-resolution image (medium priority). Once ACKed, if BBR indicates sufficient bandwidth, the application sends data for the high-resolution version (low priority), which TCP will transmit only after pending higher-priority data is cleared.

- **Supporting Data Constraints:** An application, informed by user preference, could signal via the extended API that data below a certain priority level should not be transmitted by TCP for this connection. TCP would then refrain from sending segments associated with those lower priorities.

- **Mandatory Fairness Mechanisms:** To prevent starvation and ensure equitable network resource usage:
  Intra-flow Fairness: Implement safeguards ensuring low-priority data eventually progresses (e.g., assigning a small, guaranteed fraction of the sending opportunity, priority aging).
  Inter-flow Fairness: Strictly limit the aggregate advantage conferred by prioritization. The overall flow must remain "TCP-friendly" relative to standard TCP flows (CUBIC, Reno) and fair to competing BBR flows. Congestion control modifications must prioritize stability and fairness over aggressive prioritization [cf. 31].

3.4 Refuting Anticipated Technical Objections

- **Middlebox Interference [24]:** While real, the impact might be mitigated by increasing encryption (limiting deep inspection) and focusing on the tangible benefits achievable between two cooperating, updated endpoints (client and edge server). The primary gain is sender-side scheduling.

- **Compounding BBR Fairness Issues [23, 41]:** Acknowledged. The design *must* incorporate explicit fairness controls as a primary constraint, potentially making the prioritized version *fairer* than baseline BBR under certain conditions by managing internal buffer contention more intelligently. The goal is not to make BBR more aggressive, but more semantically aware in its scheduling.

- **Priority Definition Complexity/Gaming:** This remains an application-level challenge. The TCP mechanism provides the *enforcement* capability; defining the *policy* is external. Initial deployments can use simple, coarse-grained priorities. Mechanisms to prevent gaming (e.g., browser policies) can evolve alongside adoption.
- **Interaction with Application Protocols (HTTP) and Potential Delays:** Concerns may arise about how prioritized TCP interacts with protocols like HTTP, particularly regarding potential delays or "hanging" of requests.
- **Nature of Delay:** It is necessary to understand that delaying lower-priority segments to favor higher-priority ones *within the same TCP connection* under bandwidth constraints is the *intended behavior* of this proposal. This is sender-side scheduling, distinct from traditional receiver-side TCP Head-of-Line (HoL) blocking caused by packet loss.
- **HTTP/1.1:** In sequential HTTP/1.1 requests on a persistent connection, prioritization primarily affects the delivery order of segments within a single large response. Pipelined requests (rarely used effectively) could see a later request delayed if an earlier response contains low-priority data consuming limited TCP sending capacity.
- **HTTP/2 Multiplexing:** HTTP/2 multiplexes multiple requests/responses (streams) over a single TCP connection [10] and has its own stream prioritization logic. The proposed TCP prioritization operates at a layer below HTTP/2 streams. When TCP bandwidth is limited, it will schedule segments based on the priority assigned via the socket API, regardless of which HTTP/2 stream the segment technically belongs to. This creates a potential interaction:
  - **Coordination is Key:** Ideally, the priority assigned at the TCP socket level should directly correspond to the application-level priority (e.g., derived from HTTP/2 stream weights or RFC 9218 priorities). If a resource is high priority in HTTP/2, the server application *must* signal high priority to TCP when sending its data. Consistent signaling ensures smooth coordinated operation.
  - **Potential Conflict:** If priorities conflict (e.g., HTTP/2 signals low priority, but the server mistakenly tells TCP socket the data is high priority), the TCP-level priority would likely dominate the actual transmission order when bandwidth is scarce, potentially undermining the intended HTTP/2 schedule. This serves to emphasize the need for careful implementation in the server application logic that bridges HTTP requests to TCP socket writes.
- **Mitigation via Fairness:** The mandatory intra-flow fairness mechanisms (as described in Section 3.3) are indispensable here. They ensure that even the lowest-priority data within the connection eventually makes progress, preventing indefinite stalls or "hanging" and guaranteeing eventual completion of all data transmission, albeit potentially delayed.
- **Stateless HTTP:** HTTP's stateless request-response nature is orthogonal to this. TCP provides the stateful connection over which these requests flow. The proposal modifies how data is scheduled *within* that stateful TCP connection.

Therefore, while prioritization intentionally introduces relative delays for lower-priority data, careful coordination between application-level priority signals and socket-level signals, along with robust transport-level fairness mechanisms, is necessary to prevent pathological blocking and ensure predictable behavior, especially when multiplexing protocols like HTTP/2 are used.

3.5 The Essential Role of CDNs and Edge Infrastructure

Content Delivery Networks and Edge computing platforms are natural deployment points:

- They can centralize priority assignment logic (heuristics, processing developer hints).
- They can perform necessary content transformations (e.g., generating multi-resolution image variants linked to priorities.
- They can deploy the modified TCP stack on their edge servers, impacting communication with unmodified clients, who would still benefit transparently from the prioritized sending.

3.6 Enhancing Network Efficiency and Reducing Load

Beyond reordering data to improve perceived performance, CATS provides mechanisms to enhance overall network efficiency by intelligently reducing the total volume of transmitted data. This moves the protocol from a simple scheduler to a tool for network load reduction.

- **Policy-Driven Data Omission:** CATS can directly act on application-level policies to translate high-level user intent into transport-layer bandwidth savings. This is particularly powerful when implemented at a Content Delivery Network (CDN) edge server. For example, when a user enables a "Data Saver" mode on their device, the browser can signal this preference via an HTTP request header (e.g., the ‘Save-Data: on‘ client hint). Upon receiving this hint, the CDN application can instruct its CATS-enabled transport layer to completely discard or refrain from sending any data chunks associated with priorities below a certain threshold (e.g., `PRIO_BULK_ADAPTIVE` and `PRIO_SCAVENGER`). This enforces the user's data-saving policy at the earliest possible point, preventing non-essential data from ever entering the public internet.
- **Adaptive Graceful Degradation:** The Conductor logic can use real-time network state information from the underlying congestion control algorithm (like BBR's `BtlBw` estimate) to make intelligent load-shedding decisions. This happens at the application data level, before segmentation. Under conditions of persistent, severe congestion, the scheduler could be configured with a policy to proactively discard the oldest, lowest-priority **data chunks** from its send buffer queues. This prevents the transport layer's memory from bloating with non-essential data that has a low probability of timely delivery, thus preserving network capacity and memory for more critical information. This action is a form of proactive, source-based load shedding.
- **Intelligent Delivery of Adaptive and Annotated Content:** CATS's prioritization inherently prevents the over-delivery of data that is no longer relevant to the user's focus, which is a common source of wasted bandwidth in adaptive media scenarios. By ensuring that baseline, lower-quality content is always scheduled before higher-quality enhancement data, the framework makes it easier for an application to manage and cancel pending, now-unnecessary data.

- **Preventing Over-Delivery in Feeds:** For a user scrolling quickly through a video or image feed, CATS ensures that baseline, low-quality previews are delivered first. The higher-quality, low-priority data for content that a user has already scrolled past may never be sent, and the application can more effectively cancel these pending requests from the transport buffer, saving significant bandwidth.
- **Prioritizing "Hotspots" in Video:** For long-form video, an application can use timeline metadata (e.g., YouTube's "most replayed" sections) to inform transport priority. Data chunks corresponding to these "hotspots" could be assigned a medium priority. This would allow a user scrubbing through the video to see previews of the most interesting parts load almost instantly, enhancing content discovery. In this model, the data for the immediate playback point would remain the highest priority, followed by hotspot data, and then the rest of the linear video data.

4 Low-Level Mechanism Design

This section details the preliminary low-level design for implementing CATS. This specification serves as a concrete foundation for the proposed ns-3 simulation work and future prototyping. The design is intended to be both effective and minimally invasive to the core TCP state machine in its initial form.

4.1 Extended Socket API Specification

To communicate priority from the application to the transport layer, the socket API must be extended. We propose a mechanism inspired by existing ancillary data messages (`sendmsg`/`recvmsg`) and socket options (`setsockopt`).

- **Per-Message Priority (Primary Mechanism):** The most flexible approach is to specify priority on a per-write basis. This will be modeled using a custom Tag that can be attached to packets in ns-3's application layer. In a real system, this would be analogous to using `sendmsg()` with a control message.
  - **Model:** An application preparing data to send would also create a `PriorityTag` containing a priority value.
  - **Priority Levels (Initial):** We propose five distinct priority levels for initial simulation, mapping well to the use cases:

    ```
    0: PRIO_CONTROL (e.g., C2, SSH keystrokes, XR tracking)
    1: PRIO_INTERACTIVE (e.g., critical web content, API responses)
    2: PRIO_NORMAL (e.g., normal images, non-critical data)
    3: PRIO_BULK_ADAPTIVE (e.g., adaptive video, high-res images)
    4: PRIO_SCAVENGER (e.g., background updates, logs)
    ```

- **Per-Socket Default Priority (Fallback):** An application can set a default priority for a socket. Any data written without a per-message priority tag will inherit this default.
  - **Model:** This will be modeled as a new attribute on the ns-3 `TcpSocketBase` class. In a real system, this would be a `setsockopt()` call, e.g., `setsockopt(fd, IPPROTO_TCP, TCP_PRIORITY, &prio_level, sizeof(prio_level));`.

This two-tiered approach provides both flexibility for complex applications and simplicity for basic ones.

4.2 TCP Option Format for End-to-End Priority Signaling

While the initial focus of this research is on sender-side scheduling, the TCP Option is a vital component of the complete end-to-end vision. It serves to communicate the data's semantic priority to the receiver and the network itself, enabling further optimizations beyond the sender. The proposed format is as follows:

- **Kind:** Experimental (e.g., 254, as per RFC 4727).
- **Length:** 3 bytes.
- **Info (1 byte payload):** The 3 most significant bits (MSB) encode up to 8 priority levels. The remaining 5 bits are reserved for future use.

The purpose of this explicit signaling is threefold:

1. **Informing the Receiver's TCP Stack:** A receiver's TCP stack, upon seeing a segment with a high-priority TCP Option, can make intelligent decisions. For example, it could be configured to bypass **Delayed ACK** mechanisms and send an immediate acknowledgment for critical data. This would accelerate the sender's feedback loop, allowing its congestion window to open faster and subsequent critical data to be sent sooner.
2. **Informing the Receiver's Application:** The priority information, communicated via the TCP Option, can be passed up to the receiving application through the extended socket API. This allows the receiving application (e.g., a web browser) to implement its own priority-aware processing. For instance, a browser could immediately parse and render a newly-arrived, high-priority chunk of critical CSS, even if it has a lower-priority chunk of JavaScript already in its processing queue. This extends prioritization from network delivery to application-level execution.
3. **Cooperation with Network-Level QoS:** The TCP Option provides a clean and automated link between application-semantic priority and network-level priority (DSCP). The sender's operating system can be configured with a policy: *If a TCP segment has a CATS Option indicating `PRIO_CONTROL`, automatically set the encapsulating IP packet's DSCP field to Expedited Forwarding (EF).* This creates a seamless, end-to-end QoS framework where application intent is translated into preferential treatment by network routers.

While the initial ns-3 simulations will focus on validating the sender-side scheduling benefits, the implementation of this TCP Option is a key part of the formal mechanism design and is significant for realizing the full potential of the system.

4.3 Priority-Aware TCP Send Buffer and Scheduling Algorithm

The core logic modification occurs within the TCP send buffer management and the segment transmission scheduling. We will modify the ns-3 `TcpSocketBase` send buffer logic.

It is a core tenet of this design that prioritization and scheduling decisions operate on application data *before* segmentation and sequencing. The TCP stack assigns sequence numbers sequentially based on the order in which data is selected for transmission by the CATS Conductor. This ensures that the TCP byte stream sent to the receiver remains perfectly sequential, thereby completely avoiding any receiver-side Head-of-Line blocking that would otherwise be caused by out-of-order sequence number transmission.

- **Buffer Structure:** Instead of a single FIFO buffer, the send buffer will be logically partitioned into multiple queues, one for each priority level. In ns-3, this can be implemented as five distinct `std::list` or `std::deque` objects. When the application sends data with a `PriorityTag`, the data is enqueued into the corresponding priority queue.
- **Scheduling Algorithm (Pseudocode):** The main TCP sending logic (often called ‘SendPacket‘, ‘SendPendingData‘, or similar in a TCP implementation) is modified. This function is called when the TCP state machine determines it is eligible to send new data (i.e., the send window is open, pacing allows it, etc.).

```
function schedule_and_send():
  // Pre-condition: BBR logic has determined a "send budget":
  //   can_send_bytes = min(cwnd - in_flight, pacing_allowance)

  while can_send_bytes > 0:
    segment_to_send = NULL

    // 1. Check for pending retransmissions (highest implicit priority)
    if retransmit_queue is not empty:
      segment_to_send = retransmit_queue.pop()
    else:
      // 2. Iterate through priority queues from HIGHEST to LOWEST
      for prio_level from PRIO_CONTROL to PRIO_SCAVENGER:
        if priority_queue[prio_level] has data:

          // 3. Apply Intra-flow Fairness (Starvation Prevention)
          if prio_level > PRIO_INTERACTIVE: // For non-critical data
            if not fairness_allows_tx(prio_level):
              continue // Skip this level for now

          segment_to_send = create_segment_from(priority_queue[prio_level])
          break // Found a segment to send

    if segment_to_send is NULL:
      break // No eligible data to send

    // 4. Transmit the segment
    transmit(segment_to_send)
    can_send_bytes -= segment_to_send.size

    // 5. Update fairness state
    update_fairness_state(segment_to_send.priority)
```

- **Intra-flow Fairness Mechanism (`fairness_allows_tx`):** To prevent starvation of low-priority data, a simple but effective mechanism is essential for the initial simulation. We propose a credit-based or token bucket scheme per priority level.
  - Each time a high-priority segment is sent, a small "debt" is accrued for the lower-priority queues.
  - A low-priority queue can only be serviced if it has data AND its "debt" is below a certain threshold (or it has accrued enough "credits" from periods of inactivity).
  - A simpler alternative is a timer-based override: if the data at the head of a low-priority queue has been waiting for more than a threshold time (e.g., 500ms), it is temporarily promoted to a higher priority for scheduling. This will be the initial approach for the ns-3 strawman.

4.4 Interaction with the Congestion Control Algorithm

While the proposed priority scheduling mechanism can operate on top of any standard TCP congestion control algorithm (CCA) by treating the CCA's output (the congestion window and pacing rate) as a transmission budget, its full potential is realized when deeply integrated with a model-based algorithm like BBR. The research will proceed in two phases to methodically evaluate this interaction.

Phase 1: Decoupled Scheduling (CCA as a Black Box)

The initial ns-3 implementation and evaluation will focus on a decoupled design.

- **Mechanism:**The CATS Conductor, our proposed scheduling logic (detailed in Section 4.3), will operate within the constraints calculated by an unmodified BBRv1 or BBRv2 implementation. BBR will independently model the network path and provide a "send budget" (pacing rate and `cwnd`). Our scheduler then intelligently spends this budget by sending the highest-priority available data.
- **Benefit:** This approach allows us to cleanly isolate and measure the performance gains from sender-side scheduling alone. It represents a minimally invasive modification and serves as a robust baseline that could, in principle, be applied to other CCAs like CUBIC, although the benefits may be less pronounced without BBR's pacing.

Phase 2 (Future Work): Coupled Design for a Priority-Aware BBR

Subsequent research, forming a key part of the project's long-term vision, will explore a coupled design where a bi-directional feedback loop is created between the scheduler and BBR's internal state machine. This would transform BBR from being merely network-aware to being both network and application aware. Key areas for investigation include:

- **Priority-Aware Bandwidth Probing:** The decision to enter the aggressive `PROBE_BW` state could be modulated by the priority of the data waiting to be sent. Probes could be undertaken with more confidence when high-priority, latency-sensitive data is pending, while a more conservative stance could be taken for low-priority, background traffic. This would allow the protocol to take calculated risks when the potential reward (faster delivery of critical data) is high.

- **Priority-Differentiated Reaction to Packet Loss:** BBR's reaction to packet loss could be influenced by the priority of the lost segment (known via the TCP Option or sender-side records). The loss of a `PRIO_CONTROL` segment could be interpreted as a strong signal of persistent congestion, triggering a significant rate reduction. Conversely, the loss of a `PRIO_SCAVENGER` segment might be treated as a more transient event, allowing for a faster recovery and preventing unnecessary degradation of the bottleneck bandwidth estimate.
- **Priority-Aware Startup Behavior:** The aggressiveness of BBR's initial `STARTUP` phase, where it rapidly increases its rate, could be tuned based on the priority of the initial data. A connection carrying a critical XR stream could employ a fast ramp-up to claim necessary bandwidth, while a connection for a low-priority bulk data transfer could use a slower, more network-friendly startup to avoid causing collateral packet loss to other flows.

This advanced, coupled design represents a forward-thinking evolution in congestion control, transforming the protocol from one that merely reacts to the network path to one that proactively adapts its behavior based on the semantic importance of the data it is carrying.

Limitations and Non-Goals: Packet Preemption

It is necessary to clarify that the CATS mechanism operates as a **non-preemptive scheduler**. The prioritization occurs when selecting which application data to segment and transmit at each sending opportunity. Once a segment (e.g., a large, low-priority data chunk) has been constructed and handed to the IP layer for transmission, the scheduler cannot interrupt this ongoing transmission to send a newly-arrived high-priority segment. The high-priority data must wait for the next scheduling opportunity.

True **packet preemption**, where a high-priority packet can interrupt the physical transmission of a lower-priority packet, is a more advanced mechanism being explored for ultra-low-latency networks like 5G URLLC. These schemes, sometimes called *punctured scheduling*, operate at the link layer, a layer below TCP.

CATS is therefore a complementary, end-host mechanism. It minimizes the delay before a high-priority packet *begins* transmission by ensuring it is not stuck behind lower-priority data in the sender's software queues. It does not address the serialization delay incurred while another packet is already *on the wire*. For the scope of this initial research, we focus solely on non-preemptive scheduling, as it represents a significant improvement that can be implemented by modifying only the TCP stack, without requiring specialized hardware or link-layer support.

5 Illustrative Use Cases

While enhancing web performance is a primary motivator for this research, the fundamental concept of transport-level semantic prioritization holds significant potential across diverse domains. The use cases can be categorized based on their operating environment and objectives, ranging from mission-critical tactical networks to everyday interactive applications.

5.1 High-Stakes Environments: Defense and Tactical Networks

The proposal's core concept finds a compelling and powerful application in defense and tactical communication environments. Drawing inspiration from the Internet's origins in resilient, packet-switched defense networks, this work addresses the critical challenges of modern tactical networks. These environments, often characterized as Mobile Ad-hoc Networks (MANETs), operate with constrained, lossy, and variable wireless links, yet must reliably carry heterogeneous traffic of mixed criticality [2].

Application to Satellite Communication (SatCom) Links

A critical component of modern tactical and defense networks is satellite communication. The challenge of optimizing TCP performance over these links is a fundamental problem in networking, recognized for decades due to inherent characteristics like long propagation delays and non-congestive packet loss [36, 40]. This challenge has gained renewed urgency with the large-scale deployment of Low-Earth-Orbit (LEO) satellite constellations, which introduce new complexities, including highly dynamic connectivity and significant delay variations from frequent handovers.

A recent, direct evaluation of modern congestion control schemes over these LEO networks confirms that while all protocols face degradation, TCP BBR, upon which we aim to introduce prioritization logic, is notably resilient as it handles the dynamic connectivity with only moderate performance degradation compared to loss-based (Cubic) or delay-based (Vegas) schemes [9]. This verifiable result strongly validates the choice of BBR as a robust baseline for any advanced protocol designed for modern satellite networks.

However, this research focuses on how congestion control algorithms react to the network path. It does not address the orthogonal problem of **end-host contention**: a scenario where multiple application data streams of varying mission-criticality compete for transmission over a single, precious satellite channel before they are even sent. This represents a clear research gap. For instance, a large, low-priority logistics file transfer can occupy the TCP send buffer, delaying a small, time-critical command-and-control (C2) message queued moments later.

CATS is designed to solve this specific end-host scheduling problem. It leverages BBR's effective handling of the underlying satellite path, but adds a layer of intelligence to manage what is sent. By allowing an application on a remote terminal to assign the highest priority to a C2 message or a critical sensor alert, the protocol would ensure this vital data is scheduled for transmission immediately. This moves beyond simply adapting to the path, towards intelligently managing the data sent over it, which is essential for maximizing the utility and responsiveness of constrained satellite links.

CATS Mechanism and Mission Impact

CATS can be designed to enforce mission-critical precedence at the transport layer, directly on the sending device, before data even enters the wider network.

- **Command and Control (C2) Precedence:** Tiny, latency-sensitive C2 messages (e.g., targeting updates, threat alerts, remote control signals for UAVs) can be assigned the highest priority. This ensures they are transmitted from the end-system's buffer before pending, larger data chunks from lower-priority streams, such as high-resolution video feeds or bulk logistical data.
- **Interaction with In-Network QoS:** This end-host scheduling mechanism is a powerful comple-

ment to existing in-network prioritization architectures like the IP Differentiated Services (DiffServ) framework [17], which forms the basis for military Multi-Level Precedence and Preemption (MLPP). The complete workflow could be:

1. **CATS** ensures a critical segment is sent from the host first.
2. The IP layer, guided by the segment's **CATS TCP Option** (as defined in Section 4.2), marks the packet with a high-precedence DSCP value.
3. Routers in the tactical network give the packet preferential treatment based on its DSCP mark.

This creates a true end-to-end precedence framework, from the application buffer to the final destination.

- **Enhanced Situational Awareness:** By enabling the prioritization of Blue Force Tracking updates or critical sensor alerts over less timely data, the mechanism directly contributes to a more accurate and real-time common operational picture. This is a cornerstone of modern military doctrine such as Network-Centric Warfare (NCW) [3].

5.2 Latency-Sensitive Interactive Applications

Beyond defense, the principle of preserving interactivity under load is critical for many established remote access and real-time collaboration tools that rely on TCP.

- **Remote Desktop Protocols (e.g., RDP, VNC over TCP):** User input events (keystrokes, mouse movements) are extremely latency-sensitive and demand the highest priority. Screen updates directly resulting from those actions can be prioritized next, while updates to static screen regions or background file transfers are assigned lower priorities to prevent interactive lag.
- **Interactive Shells (SSH):** For protocols like SSH [46], echoing user keystrokes and immediate command responses are critical for interactivity. Conversely, bulk data transfers over the same connection (e.g., via SFTP or displaying a large file) can be marked with lower priority to keep the shell responsive.
- **Real-time Collaboration and Messaging Systems:** In collaborative editing tools, real-time text updates can be prioritized over presence notifications, synchronization of non-visible document sections, or large embedded media file transfers.
- **Other Interactive Data Streams:** Similar benefits apply to interactive database queries (prioritizing initial results over large data set fetches) or financial data feeds (prioritizing critical market updates over auxiliary data).

5.3 Future Applications in 5G/6G Systems

The architectural shift in 5G and the forward-looking vision for 6G are defined by the need to support a diverse set of services with conflicting performance requirements on a unified infrastructure [1]. The coexistence of Enhanced Mobile Broadband (eMBB), massive Machine-Type Communications (mMTC), and Ultra-Reliable Low-Latency Communication (URLLC) creates a fundamental resource management challenge [37]. This end-host prioritization becomes particularly effective with two core 5G/6G architectural enablers, which are Network Slicing and Multi-access Edge Computing (MEC) [42]. CATS acts as the crucial "first-mile" scheduler, in that it ensures that an application's most latency-sensitive data is the first to be submitted to its designated high-performance network slice, maximizing the end-to-end performance benefits of these new architectures.

- **Extended Reality (XR) for Mobile and Aerial Platforms:** XR applications are a primary driver for 5G and future networks, imposing stringent demands for both high bandwidth and ultra-low latency. These challenges are amplified when delivering XR experiences via mobile platforms like Unmanned Aerial Vehicles (UAVs) for applications in public safety, industry, or defense [8]. An XR application running on such a platform generates a mixed-criticality data stream:
  - **Highest Priority (URLLC-like):** Device pose (head/controller tracking), user input, and real-time control signals. These are typically small, but their delivery is extremely latency-sensitive to prevent motion sickness and ensure accurate operation.
  - **Lower Priority (eMBB-like):** High-resolution textures, complex 3D models streamed from the edge, environment maps, and other non-interactive assets. These are often large but can tolerate higher latency.

  Without transport-level prioritization, the transmission of a large 3D model could block a small but critical tracking update, causing a noticeable stutter that breaks the immersive experience or compromises remote control. CATS would allow the XR application to assign the highest priority to the latency-sensitive pose data, ensuring it preempts any pending asset data in the TCP send buffer and is dispatched immediately.

- **The Tactile Internet and Remote Control:** The vision of the Tactile Internet, a cornerstone of 6G research, involves transmitting touch and actuation in real-time to control remote robotics or interact with haptic interfaces [20, 39]. This requires end-to-end latencies on the order of 1-10 milliseconds. In such a system, CATS would provide the mechanism for the application to guarantee that control and haptic signals—the essence of the "tactile" experience—are never delayed by pending video or log data at the transport layer, providing a critical component for achieving the required end-to-end performance.

5.4 Primary Motivating Use Case: Web Performance

Finally, the primary scenarios motivating this research stem from the need to improve everyday web performance, particularly under constrained network conditions.

- **Constrained News Access:** The user, such as one on a 3G connection, sees the article text load within seconds, while a low-resolution lead image appears shortly after. Web fonts, comments, and high-resolution imagery populate gradually, rather than contending equally for initial bandwidth.

- **Mobile E-commerce:** Critical path elements (product title, price, buy button) render rapidly, allowing a user to decide to purchase before all image thumbnails or related product carousels have fully loaded.

- **Image Galleries:** Low-resolution previews for all images load quickly (medium priority). Full-resolution versions (low priority) download in the background or upon user interaction.

- **Low-Data Mode:** A user enables OS-level data saving. The news site, via priority-aware TCP, transmits only the core text and minimal CSS (highest priority), actively preventing transmission of lower-priority image or script data and resulting in significant data savings.

6 Analysis of Challenges and Mitigation Strategies

Successful realization requires overcoming significant hurdles:

- **Backward Compatibility:** Ensuring seamless operation with legacy TCP stacks is non-negotiable.
  Mitigation: Adherence to RFC 793 regarding unknown options; negotiation of capabilities during handshake; initial deployment focusing on server-side benefits.

- **Standardization Effort:** Achieving IETF consensus (e.g., within TCPM WG or a new WG) is a lengthy, demanding process requiring robust technical justification and community support.
  Mitigation: Rigorous research findings, prototype data, clear performance/fairness analysis, addressing layering concerns proactively, potentially starting with Experimental RFCs.

- **Ecosystem Adoption:** Requires updates across diverse OS vendors, CDN providers, potentially browser/application developers.
  Mitigation: Demonstrating compelling benefits; targeting CDNs first; leveraging open-source implementations; providing clear APIs and documentation.

- **End-to-End QoS Limitations:** Lack of guaranteed priority handling by intermediate routers.
  Mitigation: Focus mechanism on end-host send buffer scheduling, providing benefits independent of transit network behavior.

- **Fairness Guarantees:** Ensuring fairness is particularly important when modifying complex, model-based congestion control algorithms like BBR, whose own fairness characteristics have been a subject of study [23, 41]. The introduction of an additional prioritization dimension necessitates even more rigorous fairness controls and validation, drawing lessons from prior work on specialized transport protocols [e.g., 4, 33] and on mechanisms designed specifically to ensure the fair coexistence of different transport protocols [28].
  Mitigation: Incorporate explicit fairness algorithms (e.g., weighted fair queuing principles, rate limits on priority advantage) into the core TCP modification design; validate extensively via simulation.

- **Priority Policy Definition:** Complexity and potential for misuse in assigning priorities.
  Mitigation: Separate mechanism from policy; promote best practices; potential for client-side policy enforcement (browsers); start with simple, demonstrable use cases.

- **Security Vulnerabilities:** Potential for new attack vectors.
  Mitigation: Thorough security review throughout design and standardization; extensive fuzzing and penetration testing of implementations.

- **Socio-Economic and Stakeholder Resistance:** De-prioritization of certain content types (e.g., advertisements, third-party trackers) could face significant resistance from stakeholders reliant on immediate content display for revenue (e.g., the web advertisement industry) or functionality. This is a non-technical hurdle that can impede standardization and adoption.
  Mitigation/Consideration: Acknowledge this challenge; focus initial investigations on user-centric

benefits (faster core content, data savings); frame prioritization as a mechanism that could be configured by applications/users based on policy, rather than inherently penalizing specific content types; Explore models where essential ad framework components might receive higher priority than the ad creatives themselves; standardization efforts would need to involve diverse stakeholders to find acceptable compromises or demonstrate overwhelming user benefit; Further research could quantify the impact of "core content first" on overall user engagement, which might indirectly benefit even advertising if users are less likely to abandon pages.

7 Comparative Analysis with Alternative Architectures and Techniques

The CATS framework must be evaluated not in isolation, but in the context of the evolution of internet protocols and the rich ecosystem of existing optimization techniques. Its novelty lies in its transport-native design philosophy, which presents different trade-offs compared to established approaches.

Protocol-Level Comparisons

- **HTTP/1.1 with Parallel Connections:** HTTP/1.1 is limited by strict head-of-line (HoL) blocking, forcing browsers to use multiple parallel TCP connections as a workaround for concurrency. This is fundamentally inefficient, as these connections compete with each other for bandwidth rather than cooperating, each requiring its own congestion control state and slow start phase. **CATS addresses this** by enabling fine-grained, cooperative prioritization within a single, long-lived TCP connection, providing the benefits of concurrency without the high overhead.
- **HTTP/2 over TCP [10]:** HTTP/2 solved application-level HoL blocking with streams, but it remains susceptible to **transport-layer HoL blocking**. The loss of a single TCP segment stalls the entire connection, preventing the delivery of data for all multiplexed streams until that segment is retransmitted. A lost, low-priority image packet can block a fully-arrived, high-priority CSS stream. Furthermore, the effectiveness of its stream prioritization can be limited by implementation variance across servers. **CATS mitigates this** by front-loading the TCP stream with the highest-priority data, reducing the probability that critical data is stuck behind non-critical data during a loss event. The advanced Phase 2 design of CATS could also allow it to react differently to the loss of a low-priority packet, enabling faster recovery.
- **HTTP/3 over QUIC [11, 29, 32]:** QUIC represents the current state-of-the-art, solving transport-layer HoL blocking with independent, individually flow-controlled streams. Its HTTP/3 prioritization is managed at the application layer and is highly effective. **CATS explores a different design philosophy:** a transport-native, more tightly integrated approach. The hypothesis to be tested is whether a single, coherent data stream orchestrated by a "Conductor" with direct access to BBR's real-time network model can make more efficient and responsive scheduling decisions than an application managing multiple streams based on coarser feedback. CATS is not presented as a replacement for QUIC, but as a valuable area of research into an alternative architectural trade-off, with potential benefits for the vast TCP ecosystem.

Application and Edge-Level Techniques

Beyond the transport protocol itself, CATS is complementary to several powerful optimization techniques that operate at different layers of the stack.

- **Browser Hints (‘fetchpriority‘, ‘preload‘) [44, 45]:** These hints are crucial signals that influence the browser’s internal resource discovery and fetch scheduling queues. They help the browser decide what to ask for and in what order. **CATS is complementary, not substitutive.** While browser hints control the order of HTTP requests, CATS controls the transmission order of the resulting data segments within the TCP connection itself. A high-priority fetchpriority request would ideally be paired with a high-priority CATS data transfer.
- **Service Workers [43]:** Service workers offer a powerful, programmable cache and request interception layer in the browser. They are exceptional for offline-first strategies, serving critical content from a local cache, and managing background synchronization. Their focus is on request handling and caching logic. **CATS operates at a different stage;** it optimizes the delivery of data over the network for requests that do result in a network fetch, which is a process that service workers do not directly control.
- **CDN Optimizations:** CDNs are experts in content transformation (e.g., image resizing), caching at the edge to reduce RTT, and routing traffic efficiently. They generally operate without deep, real-time insight into the TCP connection state of a specific client. **CATS provides this missing link.** It could be deployed on CDN edge servers, allowing them to pair their content optimizations with transport optimizations that are highly responsive to the real-time network conditions of each individual user.

The central research question remains: does the hypothesized benefit of CATS’s direct network awareness translate into measurable performance gains over existing solutions, sufficient to warrant the costs of modifying the transport layer? The goal is to determine if this transport-native approach can provide a unique and significant improvement to the end-to-end performance puzzle.

8 Proposed Research Methodology

A multi-faceted approach is required:

1. **Formal Mechanism Design:** Specify the TCP Option, socket API extensions, priority queue logic for the send buffer, and precise modifications to BBR’s state machine and control loops, including fairness algorithms.
2. **Simulation Environment Setup:** The primary evaluation will be conducted using a discrete-event network simulator, likely ns-3, due to its comprehensive TCP/IP modeling capabilities and flexibility.
3. **Initial Simulation Scenarios (Simple Topologies):**
    - Dumbbell topology: To analyze core prioritization behavior, interaction with congestion control (modified BBR), and impact on single-flow metrics under varying bottleneck bandwidth, RTT, and buffer sizes.

- Parking lot topology: To assess inter-flow fairness between CATS flows and standard TCP (CUBIC, BBR) flows, and among multiple CATS flows with different priority mixes.

4. **Advanced Simulation Scenarios (Representative Topologies):** Subsequent simulations will utilize more complex topologies reflecting realistic web access patterns and potentially mobile network characteristics to evaluate performance under more diverse conditions.

5. **Rigorous Fairness Analysis:** Design specific simulation scenarios and testbed experiments aimed at stressing fairness conditions (e.g., high contention, mix of flow types, persistent low-priority data) to validate the effectiveness of embedded fairness mechanisms.

6. **Performance Evaluation:** Perform direct performance and fairness comparisons against QUIC/HTTP/3 implementations under identical workloads and network conditions. Also consider, where conceptually applicable, the performance principles demonstrated by specialized datacenter transports like pFabric [4] or PIAS [7] as points of reference for what can be achieved with aggressive, context-aware transport scheduling, even if their specific mechanisms are not directly transferable to general Internet paths, to provide aspirational benchmarks for segment-level scheduling effectiveness.

7. **Complexity and Feasibility Assessment:** Analyze the implementation complexity and potential hurdles to standardization and deployment based on design and prototyping experiences.

9 Future Research Directions

Beyond the core research methodology outlined in this proposal, the CATS opens up several promising avenues for future investigation that would build upon the initial findings.

9.1 Automated Priority Assignment

While the initial simulations will rely on explicit, application-defined priorities to establish a clear performance baseline, a key challenge for widespread adoption is the automation of this process. Future work could involve designing and evaluating various automated assignment strategies to reduce the burden on developers and handle dynamically generated content.

These strategies could range in complexity:

- **Advanced Heuristics:** Developing sophisticated, rule-based heuristics that go beyond simple content-type matching. For example, a system could analyze a webpage's CSS to identify render-blocking resources or analyze the DOM structure to determine which elements are positioned *above the fold*, and assign priorities accordingly.

- **Machine Learning (ML) Models:** As a more advanced approach, an ML model could be trained for this task, particularly for complex web content. A model, potentially a Graph Neural Network (GNN) to interpret the DOM's tree structure, could be trained on features from the DOM and CSS to predict a "criticality score" for each resource. This model would be designed to run efficiently on a server or CDN edge node, analyzing page structure to generate priority metadata for CATS.

- **Hybrid Approaches:** A practical system might combine both methods, using fast heuristics for common cases and invoking a more computationally expensive ML model for complex or unknown page layouts.

Investigating the trade-offs between the accuracy of these different automated methods and their computational overhead would be a crucial area of research for making this protocol practical at scale.

9.2 Applicability to Other Transport Protocols (e.g., QUIC)

While this proposal focuses on modifying TCP due to its ubiquity, the core principles of the CATS framework are not exclusive to TCP. A significant area for future work would be to investigate how a "Conductor" mechanism could be integrated into other transport protocols, notably QUIC.

QUIC's stream multiplexing successfully solves the transport-layer head-of-line blocking problem. However, its congestion control is still a shared resource that is largely agnostic to the semantic importance of the data within the streams it manages. Future research could explore a "QUIC-Conductor" that would:

- **Inform Congestion Control:** Use priority hints from the application's streams to make QUIC's underlying congestion controller (often BBR-based) priority-aware. This would enable the advanced, Phase 2 interactions discussed earlier, such as reacting differently to the loss of a high-priority packet or using critical data for bandwidth probes.
- **Enhance Stream Scheduling:** Go beyond simple round-robin or weight-based stream scheduling by using a more sophisticated, priority-driven scheduler that also considers network state feedback to decide which stream gets the next sending opportunity.

Applying the CATS philosophy to QUIC could combine the best of both worlds: QUIC's loss-independent streams and CATS's tightly-coupled, network-aware scheduling intelligence. This represents a promising path toward a next-generation, application-aware transport protocol.

9.3 High-Performance Prototyping with eBPF

As network speeds increase, the performance of the end-host network stack itself becomes a critical factor. To validate the concept in real-world, high-performance scenarios, a future phase of this research could involve developing a Linux-based prototype using the **eBPF (Extended Berkeley Packet Filter)** framework. Instead of a deep and invasive modification of the kernel's TCP source code, an eBPF program could be attached to the TCP send-path hook. This program would implement the priority-aware scheduling logic in a way that is performant, safe, and more easily deployable than traditional kernel modules, aligning with modern trends in high-performance networking.

9.4 Interaction with Preemptive Link Layers

As discussed in Section 4.4, this proposal focuses on non-preemptive scheduling. An area for future research is the interaction between our TCP-level scheduler and emerging 5G/6G link layers that do support preemption (e.g., "punctured scheduling"). Investigating how the two schedulers can coordinate—for example, using the CATS TCP Option to signal to the link layer that a packet is eligible for preemption—could lead to a holistic, cross-layer prioritization system that minimizes latency at both the queuing and serialization stages.

10 Conclusion

This proposal advocates for a structured investigation into implementing CATS (Conductor-driven Asymmetric Transport Scheme) directly at the transport layer. The core hypothesis is that by making the transport stack aware of application-level data importance, particularly when coupled with a network-aware congestion control algorithm like BBR, we can achieve a more responsive and efficient delivery of critical content. The documented evolution of TCP to embrace performance-enhancing complexity provides a strong precedent for this line of inquiry.

The research is framed as a scientific assessment of a transport-native design philosophy for prioritization. It directly addresses known performance bottlenecks in the web stack and explores architectural trade-offs relative to state-of-the-art protocols like QUIC. The significant challenges, such as technical hurdles like ensuring fairness to practical obstacles like standardization, are acknowledged.

Ultimately, this work seeks to determine the feasibility of evolving transport protocols to be more "content-aware." The findings will either provide evidence for a new direction in TCP's evolution or inform future developments in other protocols like QUIC by demonstrating the value of tightly-coupled, application-aware scheduling. The investigation's relevance spans a wide range of critical use cases, from improving everyday web access to enabling high-performance tactical, remote access, and future 5G/6G applications. It contributes to the broader goal of creating a more performant, efficient, and user-responsive internet infrastructure.

References

[1] 3rd Generation Partnership Project (3GPP). System Architecture for the 5G System (5GS). Technical Specification TS 23.501 V19.4.0, 3GPP, jun 2025. URL `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144`. Release 19.

[2] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002. doi: 10.1109/MCOM.2002.1024422.

[3] David S. Alberts, John J. Gartska, Richard E. Hayes, and David A. Signori. *Understanding Information Age Warfare*. CCRP Publication Series, Washington, D.C., 2001. ISBN 1-893723-04-6.

[4] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320566. doi: 10.1145/2486001.2486031. URL `https://doi.org/10.1145/2486001.2486031`.

[5] Mark Allman, Vern Paxson, and Ethan Blanton. TCP Congestion Control. RFC 5681, IETF, September 2009. URL `https://www.rfc-editor.org/info/rfc5681`.

[6] Amazon Web Services, Inc. Tcp bbr congestion control with amazon cloudfront | networking & content delivery, July 2019. URL `https://aws.amazon.com/blogs/networking-and-content-delivery/tcp-bbr-congestion-control-with-amazon-cloudfront/`.

[7] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-agnostic flow scheduling for commodity data centers. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, page 455–468, USA, 2015. USENIX Association. ISBN 9781931971218.

[8] Mohammed Khaled Banafaa, Ömer Pepeoğlu, Ibraheem Shayea, Abdulraqeb Alhammadi, Zaid Ahmed Shamsan, Muneef A. Razaz, Majid Alsagabi, and Sulaiman Al-Sowayan. A comprehensive survey on 5g-and-beyond networks with uavs: Applications, emerging technologies, regulatory aspects, research trends and challenges. *IEEE Access*, 12:7786–7826, 2024. doi: 10.1109/ACCESS.2023.3349208.

[9] George Barbosa, Sirapop Theeranantachai, Beichuan Zhang, and Lixia Zhang. A comparative evaluation of tcp congestion control schemes over low-earth-orbit (leo) satellite networks. In *Proceedings of the 18th Asian Internet Engineering Conference*, AINTEC '23, page 105–112, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400709395. doi: 10.1145/3630590.3630603. URL `https://doi.org/10.1145/3630590.3630603`.

[10] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, IETF, May 2015. URL `https://www.rfc-editor.org/info/rfc7540`.

[11] M. Bishop. HTTP/3. RFC 9114, IETF, June 2022. URL `https://www.rfc-editor.org/info/rfc9114`.

[12] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016. URL `http://queue.acm.org/detail.cfm?id=3022184`.

[13] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: congestion-based congestion control. *Commun. ACM*, 60(2):58–66, January 2017. ISSN 0001-0782. doi: 10.1145/3009824. URL `https://doi.org/10.1145/3009824`.

[14] Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. *SIGCOMM Comput. Commun. Rev.*, 45(4):393–406, August 2015. ISSN 0146-4833. doi: 10.1145/2829988.2787480. URL `https://doi.org/10.1145/2829988.2787480`.

[15] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 98–109, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307970. doi: 10.1145/2018436.2018448. URL `https://doi.org/10.1145/2018436.2018448`.

[16] Cloudflare. Introducing HTTP/3 Prioritization. Cloudflare Blog, January 2023. URL `https://blog.cloudflare.com/better-http-3-prioritization-for-a-faster-web/`.

[17] David L. Black and Zheng Wang and Mark A. Carlson and Walter Weiss and Elwyn B. Davies and Steven L. Blake. An Architecture for Differentiated Services. RFC 2475, December 1998. URL `https://www.rfc-editor.org/info/rfc2475`.

[18] Deloitte Digital / Google. Milliseconds Make Millions. Industry Report, 2020. URL `https://www.thinkwithgoogle.com/_qs/documents/9757/Milliseconds_Make_Millions_report_hQYAbZJ.pdf`.

[19] Fahad R. Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. Decentralized task-aware scheduling for data center networks. *SIGCOMM Comput. Commun. Rev.*, 44(4):431–442, August 2014. ISSN 0146-4833. doi: 10.1145/2740070.2626322. URL `https://doi.org/10.1145/2740070.2626322`.

[20] Gerhard P. Fettweis. The Tactile Internet: Applications and Challenges. *IEEE Vehicular Technology Magazine*, 9(1):64–70, 2014. doi: 10.1109/MVT.2013.2295069.

[21] Google. Largest Contentful Paint (LCP). web.dev Documentation, Ongoing. URL `https://web.dev/articles/lcp`.

[22] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008. ISSN 0163-5980. doi: 10.1145/1400097.1400105. URL `https://doi.org/10.1145/1400097.1400105`.

[23] Marcel Hock, Roland Bless, and Martina Zitterbart. TCP BBR in the Wild: Large-Scale Measurement and Analysis. In *Proceedings of the Internet Measurement Conference (IMC '19)*, pages 262–275, 2019. doi: 10.1145/3355369.3355573.

[24] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, page 181–194, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310130. doi: 10.1145/2068816.2068834. URL `https://doi.org/10.1145/2068816.2068834`.

[25] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *SIGCOMM Comput. Commun. Rev.*, 42(4):127–138, August 2012. ISSN 0146-4833. doi: 10.1145/2377677.2377710. URL `https://doi.org/10.1145/2377677.2377710`.

[26] Toke Høiland-Jørgensen, Dave Täht, and Jonathan Morton. Piece of cake: A comprehensive queue management solution for home gateways. In *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 37–42, 2018. doi: 10.1109/LANMAN.2018.8475045.

[27] IETF QUIC Working Group. Priority in QUIC Transport · Issue #104 · quicwg/base-drafts. GitHub Issue Discussion, 2016-2017. URL `https://github.com/quicwg/base-drafts/issues/104`.

[28] Syed Mohammad Irteza, Adnan Ahmed, Sana Farrukh, Babar Naveed Memon, and Ihsan Ayyub Qazi. On the coexistence of transport protocols in data centers. In *2014 IEEE International Conference on Communications (ICC)*, pages 3203–3208, 2014. doi: 10.1109/ICC.2014.6883814.

[29] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, IETF, May 2021. URL `https://www.rfc-editor.org/info/rfc9000`.

[30] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912799. doi: 10.1145/52324.52356. URL `https://doi.org/10.1145/52324.52356`.

[31] Raj K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984. URL `https://www.cs.wustl.edu/~jain/papers/ftp/fairness.pdf`.

[32] M. Kühlewind and S. M. B. Priorities. Extensible Prioritization Scheme for HTTP. RFC 9218, IETF, June 2022. URL `https://www.rfc-editor.org/info/rfc9218`.

[33] Ali Munir, Ghufran Baig, Syed Mohammad Irteza, Ihsan Ayyub Qazi, Alex X. Liu, and Fahad Rafique Dogar. Pase: Synthesizing existing transport strategies for near-optimal data center transport. *IEEE/ACM Transactions on Networking*, 25(1):320–334, 2017. doi: 10.1109/TNET.2016.2586508.

[34] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, IETF, December 1998. URL `https://www.rfc-editor.org/info/rfc2474`.

[35] Kathleen Nichols, Van Jacobson, Andrew McGregor, and Jana Iyengar. Controlled Delay Active Queue Management. RFC 8289, January 2018. URL `https://www.rfc-editor.org/info/rfc8289`.

[36] C. Partridge and T.J. Shepard. Tcp/ip performance over satellite links. *IEEE Network*, 11(5):44–49, 1997. doi: 10.1109/65.620521.

[37] Parvez, Imtiaz and Rahmati, Ali and Guvenc, Ismail and Sarwat, Arif I. and Dai, Huaiyu. A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions. *Commun. Surveys Tuts.*, 20(4):3098–3130, October 2018. ISSN 1553-877X. doi: 10.1109/COMST.2018.2841349. URL `https://doi.org/10.1109/COMST.2018.2841349`.

[38] Jon Postel. Transmission Control Protocol. RFC 793, IETF, September 1981. URL `https://www.rfc-editor.org/info/rfc793`.

[39] Walid Saad, Mehdi Bennis, and Mingzhe Chen. A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems. *IEEE Network*, 34(3):134–142, 2020. doi: 10.1109/MNET.001.1900287.

[40] Luis A. Sanchez, Mark Allman, and Dr. Dan Glover. Enhancing TCP Over Satellite Channels using Standard Mechanisms. RFC 2488, January 1999. URL `https://www.rfc-editor.org/info/rfc2488`.

[41] Kanon Sasaki, Masato Hanai, Kouto Miyazawa, Aki Kobayashi, Naoki Oda, and Saneyasu Yamaguchi. TCP Fairness Among Modern TCP Congestion Control Algorithms Including TCP BBR. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–4, 2018. doi: 10.1109/CloudNet.2018.8549505.

[42] Taleb, Tarik and Samdanis, Konstantinos and Mada, Badr and Flinck, Hannu and Dutta, Sunny and Sabella, Dario. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017. doi: 10.1109/COMST.2017.2705720.

[43] W3C Service Worker Working Group. Service Workers 1. W3C Recommendation, December 2022. URL `https://www.w3.org/TR/service-workers-1/`.

[44] W3C Web Incubator Community Group (WICG) / WHATWG. Priority Hints. Specification Draft, Ongoing. URL `https://wicg.github.io/priority-hints/`.

[45] W3C Web Performance Working Group. Resource Hints. W3C Recommendation, March 2023. URL `https://www.w3.org/TR/resource-hints/`.

[46] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, IETF, January 2006. URL `https://www.rfc-editor.org/info/rfc4251`.

[47] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980. doi: 10.1109/TCOM.1980.1094702.