Indian Journal of Computer Science and Technology
https://www.doi.org/10.59256/indjcst.20250403044
Volume 4, Issue3 (September-December 2025), PP: 276-281.
www.indjcst.com
ISSN No: 2583-5300

Review of TCP Versions Using Selective Acknowledgment

Bhavika M. Gambhava[1*], Ashish K. Gor[2#], C. K. Bhensdadia[3$], Nikhil J. Kothari[4%]
[1, 2, 3] Department of Computer Engineering, Dharmsinh Desai University, Gujarat, India.
[4]Department of Electronics & Communication, Dharmsinh Desai University, Gujarat, India.

To Cite this Article: *Bhavika M. Gambhava[1*], Ashish K. Gor[2#], C. K. Bhensdadia[3$], Nikhil J. Kothari[4%], "Review of TCP Versions Using Selective Acknowledgment", Indian Journal of Computer Science and Technology, Volume 04, Issue 03 (September-December 2025), PP: 276-281.*

Abstract: *Selective Acknowledgment (SACK) represents a critical enhancement to the Transmission Control Protocol (TCP) that significantly improves performance in networks experiencing packet loss. Unlike traditional cumulative acknowledgments, SACK allows receivers to inform senders about all successfully received segments, enabling more efficient retransmission strategies. This survey examines various TCP versions and variants that incorporate SACK mechanisms, analyzing their congestion control algorithms, performance characteristics, and suitability for different network environments, particularly in heterogeneous wired-cum-wireless networks. The evolution of SACK-based TCP variants demonstrates a progressive refinement in addressing the challenges posed by random errors in wireless links while maintaining robust congestion control in wired networks. Through comprehensive analysis of established and emerging TCP variants, this survey illuminates the path toward more adaptive and intelligent transport protocols capable of meeting the demands of modern heterogeneous network infrastructures.*

Key Word: *Selective Acknowledgement, Congestion Control, TCP, Transport Protocol, Internet.*

I. INTRODUCTION

1.1 Background and Motivation

The Transmission Control Protocol has served as the backbone of reliable data transmission over the Internet since its inception, providing end-to-end reliability, flow control, and congestion management for countless applications [1] . From its early days connecting research institutions to its current role supporting billions of devices worldwide, TCP has evolved continuously to address emerging challenges. The protocol's fundamental design, rooted in the pioneering work of the 1980s, established principles of congestion control that remain relevant today [23]. However, the networking landscape has transformed dramatically since TCP's initial standardization, presenting new challenges that demand innovative solutions.

Traditional TCP implementations relied on cumulative acknowledgments, where the receiver acknowledges only the highest in-order byte received. While this approach proved sufficient for early Internet applications running over relatively stable wired links, it revealed significant limitations when faced with more complex loss scenarios. When multiple packets are lost within a single transmission window, the sender receives limited information about which specific segments failed to arrive. This informational gap forces the sender into conservative recovery strategies, often resulting in unnecessary retransmissions and prolonged recovery periods. The inefficiency becomes particularly pronounced in networks with high bandwidth-delay products, where the cost of conservative behavior translates directly into substantial throughput degradation.

Congestion control has been one of the prominent research areas as it decides the speed and efficiency of data [16] [17] [22]. The integration of wireless links, both terrestrial and satellite, with traditional wired infrastructure has introduced additional complexity to the congestion control challenge. Wireless networks exhibit fundamentally different loss characteristics compared to their wired counterparts. While packet losses in wired networks typically signal network congestion resulting from buffer overflows at intermediate routers, wireless links experience packet corruption due to channel errors, interference, fading, and mobility-related phenomena. These random errors occur independently of network load and available bandwidth. Traditional TCP implementations, designed with the implicit assumption that packet loss indicates congestion, respond to all losses by reducing transmission rates. This conservative behavior, while appropriate for congestion-induced losses, proves counterproductive when applied to random errors in wireless channels, leading to severe underutilization of available bandwidth.

1.2 The Evolution and Necessity of SACK

The inability of traditional TCP to cope efficiently with multiple packet losses led to the introduction of Selective Acknowledgment (SACK), which significantly improves information exchange between communicating endpoints. Unlike cumulative acknowledgments that reveal only the highest in-order byte received, SACK allows the receiver to report non-contiguous blocks of successfully delivered data[2][3]. This detailed feedback enables the sender to retransmit only genuinely lost segments, avoiding unnecessary retransmissions and improving overall efficiency.

Beyond efficiency, SACK enables faster and more stable recovery when multiple losses occur within a single window [4]. By supporting selective retransmission without relying on timeouts, SACK helps TCP maintain forward progress, a property that is especially important in high-speed networks where timeouts are costly. The emergence of wireless and heterogeneous networks has further highlighted SACK's importance. Wireless links exhibit frequent out-of-order delivery and random losses, conditions under which SACK provides critical insight into receiver state. As a result, SACK has become a key enabler for advanced control and loss differentiation mechanisms in modern TCP variants [5].

1.3 SACK Mechanism: Technical Foundation

The Selective Acknowledgment (SACK) mechanism enhances TCP reliability by allowing receivers to report non-contiguous data that has arrived successfully. SACK information is carried in TCP options within acknowledgment packets and is enabled during connection setup using the SACK-permitted option. Once negotiated, the receiver can include SACK blocks that specify ranges of sequence numbers already received but not yet cumulatively acknowledged. Each block identifies a contiguous segment using left and right sequence edges.

On the sender side, SACK requires maintaining a scoreboard that records the status of transmitted segments. Incoming SACK information updates this structure, helping the sender identify missing data and retransmit only the necessary segments. The sender must also account for challenges such as out-of-order acknowledgments, lost ACKs, and SACK reneging. SACK works alongside TCP congestion control rather than replacing it. By improving loss detection and recovery, it reduces unnecessary retransmissions while remaining compatible with both classic and modern congestion control algorithms.

II.TCP VARIANTS WITH SACK SUPPORT

2.1 TCP Reno with SACK: Foundation and Limitations

TCP Reno marked a major step forward from earlier TCP variants such as Tahoe by introducing fast retransmit and fast recovery, reducing reliance on timeouts [6]. Like Tahoe, Reno still treats packet loss as a sign of congestion and responds by reducing the congestion window, but it avoids dropping back to slow start after every loss. Adding SACK to Reno improves loss recovery by allowing the sender to identify and retransmit multiple lost packets within a single round-trip time. However, Reno with SACK inherits Tahoe's core limitation: it cannot distinguish congestion losses from wireless errors, leading to unnecessary rate reduction and inefficient use of wireless links, especially in mixed wired–wireless networks.

2.2 TCP NewReno with SACK: Enhanced Recovery Mechanisms

TCP NewReno improves upon Reno by handling multiple packet losses more effectively during fast recovery [7]. Its key enhancement is the treatment of partial acknowledgments: instead of exiting fast recovery early, NewReno stays in recovery until all data outstanding at the time of loss is acknowledged, avoiding costly timeouts. When combined with SACK, recovery becomes faster, as the sender can identify and retransmit several lost segments within a single round-trip time rather than one per loss. However, like earlier TCP variants, NewReno with SACK still interprets all losses as congestion, leading to unnecessary rate reduction and underutilization of wireless links where losses are often error-driven rather than congestion-related.

2.3 Adaptive Flow Control and Delayed Fast Recovery

Adaptive Flow Control (AFC) TCP was proposed to better handle losses over wireless links, where errors are often random rather than congestion-driven [8]. Building on delayed fast recovery, AFC avoids blindly reducing the congestion window after every loss. Instead, it estimates congestion severity using the displacement between consecutive loss events, measured as the amount of successful data transfer between losses. Frequent, closely spaced losses indicate heavy congestion and trigger stronger flow control, while widely spaced losses suggest mild congestion or random errors and result in a gentler response. Simulation studies showed that AFC improves throughput over wireless links while preserving fairness in wired networks, though effective performance depends on careful parameter tuning for different network conditions.

2.4 Retransmission Timeout Analysis and Channel Noise Impact

Retransmission timeouts (RTOs) pose a serious challenge to TCP performance, especially in wireless networks. An RTO occurs when a retransmitted packet is also lost, forcing TCP to assume widespread loss and drastically reduce its sending rate. The sender resets the congestion window to one segment, backs off the timer, and enters slow start, often leaving the link idle for many round-trip times. Research showed that when RTOs are triggered by wireless channel errors rather than congestion, this response is unnecessarily harsh [9]. Mathematical and simulation studies revealed that large congestion windows can lead to long idle periods, wasting significant bandwidth. AFC-based approaches address this by distinguishing error-induced timeouts from congestion, allowing limited transmission during suspected channel errors and preserving throughput without sacrificing stability.

2.5 TCP Vegas with SACK: Proactive Congestion Avoidance

TCP Vegas takes a proactive, delay-based approach to congestion control, unlike traditional loss-based TCP variants [10]. Instead of waiting for packet loss, Vegas monitors changes in round-trip time (RTT) to detect early signs of congestion. It compares expected throughput, computed from the congestion window and minimum RTT, with actual throughput based on current RTT. The difference estimates queued data in the network and is compared against two thresholds, α and β, to decide whether to increase, decrease, or maintain the congestion window. SACK complements Vegas by enabling efficient recovery when losses do occur, preventing disruptive timeouts. Vegas performs well in stable networks with low delay variation, offering smooth throughput and low loss, but it can be disadvantaged when competing with more aggressive loss-based TCP flows.

2.6 TCP Westwood with SACK: Bandwidth Estimation for Wireless Networks

TCP Westwood was designed to improve TCP performance over wireless links by using bandwidth estimation rather than blind window reduction [11]. It continuously estimates available bandwidth by observing the rate of incoming acknowledgments and smoothing these measurements with a low-pass filter. After packet loss, Westwood sets the congestion window and slow start threshold based on the estimated bandwidth–delay product instead of halving the window, avoiding underutilization caused by random wireless errors. When combined with SACK, Westwood can efficiently retransmit only missing segments while preserving accurate bandwidth estimates, even during multiple losses. Experiments show that Westwood with SACK achieves higher throughput in wireless environments. However, inaccurate bandwidth estimates due to delayed acknowledgments and potential fairness issues with other TCP variants remain open challenges.

2.7 TCP BIC with SACK: Binary Search for High-Speed Networks

Binary Increase Congestion Control (BIC) TCP was designed for high-speed networks with large bandwidth-delay products, where traditional TCP recovers too slowly after congestion [12]. Instead of linear additive increase, BIC uses a binary search strategy to quickly return to the optimal congestion window. After a loss, it records the reduced window as the minimum and the loss point as the maximum, then probes between them. Large steps are used when far from the target, and smaller steps near the midpoint, ensuring fast yet stable convergence. If conditions improve beyond the previous maximum, BIC allows further controlled growth.

SACK plays a critical role in BIC's effectiveness. Rapid window growth can cause multiple packet losses, and SACK enables fast identification and retransmission of all lost segments within a single RTT. Experiments show that BIC with SACK achieves high throughput and better RTT fairness in high-capacity networks. However, its aggressive behavior can disadvantage traditional TCP flows, leading to the development of TCP CUBIC as a fairer refinement.

2.8 TCP CUBIC with SACK: The Modern Standard

TCP CUBIC is the dominant congestion control algorithm in modern operating systems and is the default in Linux [13]. It improves upon BIC by using a cubic window growth function that depends only on the time since the last congestion event, making it largely independent of RTT and fairer to flows with different delays. The cubic function allows rapid window growth when far below the previous maximum, slows down near the earlier congestion point for stability, and accelerates again when probing for additional capacity.

SACK plays a vital role in CUBIC's performance. CUBIC's aggressive probing can trigger multiple losses, and SACK enables fast, selective retransmission within a single RTT, avoiding long recovery periods. In practice, CUBIC delivers high throughput in high-speed networks while remaining well behaved in typical broadband environments. Its widespread deployment has validated its robustness, while ongoing research continues to refine CUBIC for data centers and wireless networks.

2.9 TCP Compound: Hybrid Loss and Delay-Based Control

TCP Compound (CTCP) combines loss-based and delay-based congestion control to balance high throughput with low delay [14]. It maintains two window components: a base window that follows traditional loss-based TCP rules and ensures fairness with other flows, and a delay window that adjusts according to queuing delay. The total sending window is the sum of both, allowing CTCP to be aggressive when the network is lightly loaded and cautious as delays grow. SACK improves CTCP's efficiency during loss recovery by enabling selective retransmissions without severely disturbing delay measurements. Evaluations show that CTCP performs well in high-speed, lightly contended networks, quickly utilizing available capacity. However, in networks with heavy cross-traffic or fluctuating delays, interactions between the two window components can cause instability and complicate implementation.

2.10 TCP Illinois with SACK: Delay-Modulated Aggressiveness

TCP Illinois blends loss and delay information by adapting traditional TCP behavior based on measured queuing delay, rather than using separate window components [15]. It estimates queuing delay as the difference between current RTT and the minimum RTT, then adjusts the increase (α) and decrease (β) parameters accordingly. With low delay, Illinois increases the window more aggressively and reduces it less after loss; with high delay, it becomes conservative. SACK is essential when Illinois operates aggressively, as rapid window growth can cause multiple losses, which SACK helps recover from efficiently. Experiments show Illinois achieves high throughput in high-speed and variable-load networks. However, its reliance on delay measurements makes it sensitive to RTT variations unrelated to congestion and requires careful tuning of delay thresholds.

2.11 Discrete TCP: Differentiating Congestion Control Phases

Discrete TCP (DTCP) is motivated by the observation that TCP's two congestion control phases—slow start and congestion avoidance—serve different goals but are traditionally treated the same after packet loss[16]. DTCP introduces a phase- aware response to losses. When loss occurs during slow start, DTCP interprets it as discovery of network capacity and reduces the congestion window less aggressively, using the loss point to set more informed parameters. During congestion avoidance, DTCP adapts its window reduction based on context, such as recent loss history and the relation between the congestion window and slow start threshold, instead of blindly halving the window.

SACK enhances DTCP by revealing loss patterns. Isolated losses suggest random errors and trigger mild responses, while clustered losses indicate congestion and prompt stronger reduction. Simulations show DTCP improves throughput in wired–wireless networks while preserving stability and fairness, though accurate loss classification remains a key challenge.

2.12 Composite TCP: Loss Type Differentiation Through RTT and SACK

Composite TCP is designed to distinguish congestion losses from random wireless errors in heterogeneous networks [17]. It combines RTT measurements with SACK information to classify losses and respond appropriately. When a loss occurs, the protocol compares the current RTT with the smoothed RTT (SRTT) and examines SACK blocks to determine whether the loss is isolated or clustered. A single loss with normal RTT is treated as a random error, triggering selective retransmission without reducing the congestion window. Multiple losses or increased RTT indicate congestion, prompting standard window reduction and fast recovery. This dual response prevents unnecessary rate reduction over wireless links while preserving fairness in wired networks. Simulation studies show that Composite TCP significantly improves throughput in wired–wireless scenarios, though its effectiveness depends on accurate RTT tracking and reliable interpretation of SACK-based loss patterns.

III. COMPARATIVE ANALYSIS OF SACK-ENABLED TCP VARIANTS

3.1 Congestion Control Philosophies and Mechanisms

TCP variants that support SACK reflect different ways of interpreting network conditions and controlling congestion. Classical protocols such as Reno and New Reno follow a loss-based philosophy, where packet loss is treated as the main indicator of congestion. Their response is conservative: the congestion window is reduced multiplicatively whenever loss is detected. While SACK improves their ability to recover efficiently from multiple losses, it does not change this fundamental behavior, and all losses are still treated uniformly.

Delay-based approaches, including TCP Vegas and the delay components of Compound TCP and Illinois, adopt a more proactive view. They interpret rising queuing delay as an early warning of congestion and adjust sending rates before packet loss occurs. This allows better utilization with lower delay, but also makes these protocols sensitive to RTT variations caused by non-congestion factors such as route changes or wireless link dynamics.

Bandwidth-estimation schemes like TCP Westwood follow a different path by explicitly estimating available bandwidth from acknowledgment rates. These estimates guide window adjustment after loss, aiming for more informed responses. However, accurate estimation is challenging due to ACK delays and transient effects.

Hybrid designs such as Discrete TCP and Composite TCP combine loss patterns, delay, and protocol state. By using multiple signals, they aim to distinguish congestion from random errors, offering better performance in heterogeneous networks at the cost of higher complexity.

3.2 Performance Characteristics Across Network Environments

The performance of SACK-enabled TCP variants depends strongly on the network environment and the design assumptions of each protocol. In well-provisioned wired networks, most variants perform adequately, with differences mainly in how fast they recover from losses and how efficiently they use high-capacity links. High-speed variants such as BIC and CUBIC perform particularly well in these settings, as their aggressive window growth and SACK-based recovery allow rapid bandwidth probing and quick loss recovery.

Wireless networks pose a greater challenge due to random errors. Loss-based variants like Reno and NewReno with SACK still reduce their sending rates after each loss, leading to chronic underutilization as error rates rise. TCP Westwood improves performance by adjusting its window using bandwidth estimates, but it still reacts to all losses, limiting gains under heavy errors.

Protocols that differentiate loss types, especially Composite TCP, perform best in wireless and mixed networks. By avoiding congestion control for random errors, they achieve much higher throughput, often exceeding traditional TCP by large margins. In complex heterogeneous paths, adaptive variants such as Compound TCP, Illinois, and Composite TCP show the greatest promise, though perfect adaptation remains difficult.

3.3 Fairness Considerations and Protocol Interactions

Fairness is a central issue in congestion control, both among flows using the same protocol and when different TCP variants compete. Traditional protocols such as Reno and NewReno achieve good fairness through the additive increase, multiplicative decrease (AIMD) principle, and the use of SACK does not change this behavior since it only improves loss recovery. In contrast, high-speed variants like BIC and CUBIC raise fairness concerns in mixed environments, as their faster window growth can dominate Reno flows, especially on high-bandwidth paths. Delay-based protocols such as Vegas face the opposite problem, often yielding bandwidth to more aggressive loss-based flows and suffering poor performance. Hybrid approaches like Compound TCP try to balance competitiveness and delay sensitivity. RTT fairness is another challenge, since classic TCP favors short-RTT flows. CUBIC improves this by making window growth largely RTT-independent, though complete RTT fairness remains difficult to achieve.

3.4 Implementation Complexity and Deployment Considerations

The implementation complexity of SACK-enabled TCP varies widely across variants. Traditional protocols like Reno and NewReno add moderate complexity, requiring scoreboards to track segments, logic to process SACK blocks, and modifications to retransmission and window management. Challenges include handling SACK reneging, out-of-order SACKs, and interactions with options like timestamps. More advanced variants increase complexity: Westwood needs accurate bandwidth estimation despite ACK compression or delays, while delay-based protocols like Vegas require precise RTT tracking and integration with SACK- based recovery. Hybrid protocols such as Compound TCP and Illinois are the most complex, coordinating multiple congestion control components, maintaining separate state, and tuning parameters for diverse environments. Deployment

challenges include backward compatibility, incremental deployment, and sensitivity to network conditions, particularly for variants relying on delay measurements. Operational monitoring and debugging are also more difficult due to the intricate internal logic and multi-source decision-making.

IV.SACK IMPLEMENTATION CONSIDERATIONS

Implementing SACK at the TCP sender requires effective management of a scoreboard that tracks sent, acknowledged, and missing segments. The sender updates this structure using cumulative acknowledgments and SACK blocks to identify gaps and infer losses. In high-speed networks, large numbers of in-flight segments make efficiency critical; optimized data structures are needed to avoid excessive CPU overhead and to handle sequence number wraparound correctly. SACK-based retransmission must balance speed and accuracy, prioritizing confirmed losses while respecting the congestion window. During recovery, the sender manages transitions between normal transmission and fast recovery, handling partial acknowledgments and new loss information. Careful coordination between SACK signals and duplicate acknowledgment logic is essential for timely and stable loss recovery.

4.1 Receiver-Side Requirements and SACK Block Generation

In SACK-enabled TCP, the receiver reports out-of-order data using SACK blocks in acknowledgment packets. Because the TCP option space is limited, the receiver can usually include only three or four blocks, so it must prioritize which gaps to report. Recently received segments are reported first, as they are most useful for sender retransmission decisions. The receiver must maintain consistency across acknowledgments and manage buffers carefully, since dropping buffered data requires withdrawing previously reported SACK information, a situation known as SACK reneging. Acknowledgment timing is also important: many implementations send immediate ACKs for out-of-order segments to deliver SACK information quickly, improving loss recovery at the cost of slightly higher ACK traffic.

4.2 Overhead Analysis and Performance Trade-offs

SACK introduces overhead in bandwidth, processing, and memory. Each SACK block adds about 10 bytes to the TCP header, and with multiple blocks, acknowledgment packets can grow significantly, increasing reverse-path bandwidth usage, especially during out-of-order delivery. Processing overhead also rises: receivers must identify and encode non-contiguous data blocks, while senders must parse SACK options, maintain scoreboards, and make selective retransmission decisions. Memory usage increases mainly at the sender, which must track all outstanding segments, and at the receiver, which buffers out-of-order data. Despite these costs, SACK usually delivers clear performance gains in lossy or high–bandwidth-delay networks by enabling faster, more efficient recovery from multiple losses.

4.3 Compatibility, Interoperability, and Deployment

SACK was designed with strong backward compatibility. During connection setup, endpoints negotiate SACK support using the SACK-permitted option; if either side lacks support, TCP falls back to cumulative acknowledgments. This enables smooth, incremental deployment. Interoperability across implementations generally works well, though differences in SACK block generation, prioritization, and retransmission strategies can affect performance. Middleboxes pose a larger challenge, as some may strip or alter TCP options, reducing or eliminating SACK benefits without breaking connectivity. Despite such issues, SACK is now widely deployed and proven reliable across diverse networks. Long-term deployment has revealed edge cases, leading to ongoing refinements and extensions of SACK for modern scenarios such as multipath TCP and high-delay satellite links.

V.FUTURE DIRECTIONS AND RESEARCH OPPORTUNITIES

Machine learning offers new possibilities for SACK-based TCP by enabling adaptive congestion control that learns from network behavior [21]. ML models can improve loss classification beyond simple heuristics, predict congestion before losses occur, and dynamically tune parameters for diverse environments [19]. However, challenges remain around training data, stability, fairness, and safe deployment.

Beyond ML, richer loss differentiation is needed. Instead of a binary view of congestion versus random errors, future schemes may use multi-metric classification combining RTT, SACK patterns, jitter, ECN, and even limited cross-layer hints, especially in wireless networks. The key challenge is achieving accuracy without excessive complexity.

Emerging networks such as 5G, edge computing, and future 6G systems demand further evolution [18]. Network slicing, split TCP at the edge, and heterogeneous paths require flexible, context-aware SACK strategies. Multipath TCP amplifies these needs, as different subflows may experience very different loss behaviors, calling for path-specific SACK-based control.

Finally, IoT environments introduce constraints on energy, memory, and processing [20]. Lightweight and energy-aware SACK variants, possibly enabled selectively, can balance performance with resource limits. Across all scenarios, SACK remains a core building block for adaptive, future-proof transport protocols.

VI.CONCLUSION

The evolution of SACK-enabled TCP variants reflects sustained efforts to keep transport protocols effective across diverse and demanding networks. From SACK integration in New Reno to advanced loss differentiation in Composite TCP, this progression confirms SACK's central role in efficient loss recovery, especially under multiple packet losses. Its widespread adoption—most notably through CUBIC—demonstrates both practicality and clear performance benefits. This survey highlights three key insights. First, SACK consistently improves performance across all congestion control philosophies by providing richer feedback for selective retransmission. Second, accurate loss differentiation remains critical in wireless and heterogeneous networks, where random errors still cause underutilization. While hybrid approaches show promise, perfect classification remains

challenging. Third, no single congestion control strategy suits all environments; different networks and applications demand different behaviors. Practically, CUBIC with SACK is a strong default for general Internet use, while wireless, satellite, and specialized environments benefit from loss-aware or adaptive variants. For researchers, opportunities remain in machine learning–assisted classification, multi-metric decision-making, and optimization for 5G, IoT, and multipath transport.

References

1. Postel, J. (1981). "Transmission Control Protocol." RFC 793, Internet Engineering Task Force.
2. Mathis, M., Mahdavi, J., Floyd, S., & Romanow, A. (1996). "TCP Selective Acknowledgment Options." RFC 2018, Internet Engineering Task Force.
3. Floyd, S., Mahdavi, J., Mathis, M., & Podolsky, M. (2000). RFC2883: An extension to the selective acknowledgement (SACK) option for TCP.
4. Fall, K., & Floyd, S. (1996). "Simulation-based comparisons of Tahoe, Reno, and SACK TCP." ACM SIGCOMM Computer Communication Review, 26(3), 5- 21.
5. Allman, M., Paxson, V., & Blanton, E. (2009). "TCP congestion control." RFC 5681, Internet Engineering Task Force.
6. Padhye, J., Firoiu, V., Towsley, D. F., & Kurose, J. F. (2002). Modeling TCP Reno performance: a simple model and its empirical validation. *IEEE/ACM transactions on Networking*, *8*(2), 133-145.
7. Floyd, S., Henderson, T., & Gurtov, A. (2004). "The NewReno modification to TCP's fast recovery algorithm." RFC 3782, Internet Engineering Task Force.
8. Kothari, N., Gambhava, B., & Dasgupta, K. (2007). "Adaptive flow control: An extension to delayed fast recovery." 15th International Conference on Advanced Computing and Communications (ADCOM 2007).
9. Gambhava, B., Kothari, N. & Dasgupta, K. (2010). "Analysis of RTO Caused by Retransmission Loss to Combat Channel Noise." International Journal of Computer Applications.
10. Brakmo, L. S., & Peterson, L. L. (2002). "TCP Vegas: End to end congestion avoidance on a global Internet." IEEE Journal on Selected areas in Communications, 13(8), 1465-1480.
11. Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. Y., & Wang, R. (2002). "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links." Proceedings of the 7th annual international conference on Mobile computing and networking, 287-297.
12. Xu, L., Harfoush, K., & Rhee, I. (2004). "Binary increase congestion control (BIC) for fast long-distance networks." IEEE INFOCOM 2004. [13]Ha, S., Rhee, I., & Xu, L. (2008). "CUBIC: a new TCP-friendly high-speed TCP variant." ACM SIGOPS Operating Systems Review, 42(5), 64-74.
13. Tan, K., Song, J., Zhang, Q., & Sridharan, M. (2006). "A compound TCP approach for high-speed and long distance networks." Proceedings of IEEE INFOCOM.
14. Liu, S., Başar, T., & Srikant, R. (2008). "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks." Performance Evaluation, 65(6-7), 417-440.
15. Gambhava, B., & Bhensdadia, C. (2018). "Discrete TCP: Differentiating Slow Start and Congestion Avoidance." International Journal of Intelligent Engineering & Systems.
16. Gambhava, B., & Bhensdadia, C. (2023). "Mathematical modelling of packet transmission during reclamation period in NewReno TCP and CTCP" International Journal of Internet Protocol Technology.
17. Lorincz, J., Klarin, Z., & Ožegović, J. (2021). A comprehensive overview of TCP congestion control in 5G networks: Research challenges and future perspectives. *Sensors*, *21*(13), 4510.
18. Zhou, J., Qiu, X., Li, Z., Li, Q., Tyson, G., Duan, J., & Wu, Q. (2022). A machine learning-based framework for dynamic selection of congestion control algorithms. *IEEE/ACM Transactions on Networking*, *31*(4), 1566-1581.
19. Hasan, H. H., & Alisa, Z. T. (2023). Effective IoT congestion control algorithm. *Future Internet*, *15*(4), 136.
20. Sacco, A., Flocco, M., Esposito, F., & Marchetto, G. (2021, May). Owl: Congestion control with partially invisible networks via reinforcement learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications* (pp. 1-10)
21. Shi, H., & Wang, J. (2023). Intelligent TCP congestion control policy optimization. *Applied Sciences*, *13*(11), 6644.
22. Allman, M., Paxson, V., & Blanton, E. (2009). *TCP congestion control* (RFC 5681).