Introducing a New TCP Variant for UAV Networks Following Comparative Simulations

George Amponis[a,b], Thomas Lagkas[b], Konstantinos Tsiknas[b], Panagiotis Radoglou-Grammatikis[a] and Panagiotis Sarigiannidis[c,*]

[a]Department of Research and Development, K3Y Ltd., Sofia, 1612, Bulgaria
[b]Department of Computer Science, International Hellenic University, Kavala Campus, 65404, Greece
[c]Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, 50100, Greece

ARTICLE INFO

Keywords:
TCP
Congestion Control
Drone Swarms
Flying Ad Hoc Networks

ABSTRACT

The Transmission Control Protocol (TCP) is a reliable, connection oriented, congestion control mechanism, currently utilized the majority of both wired and wireless networks at the transport layer. An important function of TCP is the network congestion control mechanism; it governs the packet transmission rate and us enables the protocol to respond to congestion signals. Considering the wide spectrum of requirements stemming from unique network and channel characteristics, there exist numerous variants of TCP. While is commonly utilized in applications requiring reliable and ordered reception of packets, the standard TCP congestion control mechanism demonstrates poor performance in high-mobility wireless networking scenarios, as high mobility implies unreliable radio links and consecutive re-transmissions. In this paper we performed a comparative analysis of a spectrum of TCP variants, with the ultimate goal of deriving best practices to support real-time, yet reliable communication in high-mobility scenarios, with a special focus on aerial mobile ad hoc networks composed of Unmanned Aerial Vehicles (UAVs). Following the findings of this simulation evaluation, we introduce a new TCP variant for Flying Ad hoc Networks (FANETs), named Swarm HTCP (S-HTCP), which is shown to outperform the other variants in such network conditions.

1. Introduction

Ad hoc networks, and especially the ones composed of aerial nodes such as UAVs require timely transmission of (usually) real-time data and minimal overhead. For that purpose, in most use cases, developers resort to the User Datagram Protocol (UDP). UDP offers minimal transmission delay, and requires no connection setup process. When necessary, flow control or re-transmission can also be applied, (when necessary) at the application layer (1). As such, UDP is well-suited for use in real time applications, namely video and audio (surveillance/remote sensing/monitoring).

However, UDP in ad hoc networks suffers from performance degradation, which can be attributed to several factors directly associated with the core functionality of the protocol. The most important factor on which this paper focuses, is lack of a congestion control mechanism. This is particularly true for Flying Ad Hoc Networks (FANET), where node mobility is highlighted, and link states are extremely volatile.

UDP is effectively blind to the state of the overall network and is only concerned with transmitting a packet. TCP inherently implements congestion avoidance in order to address the issue of channel resource miss-allocation and reliable transmission. To do this, TCP implements a slow start phase described by Algorithm 1, a congestion avoidance phase described by Algorithm 2, and a fast recovery phase described by Algorithm 3, where applicable (2). While TCP's congestion avoidance and control mechanisms are indeed valuable for reliable data transfer, it is primarily designed for wired networks; as such, it faces performance degradation when applied to the wireless ad hoc scenario (3), where re-transmissions due to high node mobility introduce a high degree of packet loss. Given that TCP was mainly designed to recover only from congestion situations, losses of TCP segments due to errors in the transmission link are

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016941.

*Corresponding author
gamponis@k3y.bg (G. Amponis); geaboni@cs.ihu.gr (G. Amponis); tlagkas@cs.ihu.gr (T. Lagkas); ktsik@emt.ihu.gr (K. Tsiknas); pradoglou@k3y.bg (P. Radoglou-Grammatikis); psarigiannidis@uowm.gr (P. Sarigiannidis)
ORCID(s): 0000-0001-6411-0485 (G. Amponis); 0000-0002-0749-9794 (T. Lagkas); 0000-0001-9698-1285 (K. Tsiknas); 0000-0003-1605-9413 (P. Radoglou-Grammatikis); 0000-0001-6042-0355 (P. Sarigiannidis)

Algorithm 1 TCP Slow Start

Input: $(cwnd, ssthresh, congestion\_event(bool))$
Output: $cwnd$
1: $cwnd \leftarrow 1$
2: **for** $(everyACK)$ **do**
3: $cwnd++$
4: **if** $(congestion\_event$ **or** $(cwnd > ssthresh))$ **then**
5: $tcpCongestionAvoidance$
6: **end if**
7: **end for**

Algorithm 2 TCP Congestion Avoidance

Input: $(cwnd, ssthresh, timeout(bool), dupACK$
Output: $cwnd$
1: **for** $(everyACK)$ **do**
2: $cwnd = +increment\_val$
3: **if** $(timeout$ **or** $(dupACK \geq 3))$ **then**
4: $tcpFastRecovery$
5: **end if**
6: **end for**

Algorithm 3 TCP Fast Recovery

Input: $(cwnd, ssthresh)$
Output: $cwnd$
1: $cwnd = ssthresh/2$
2: $retransmit$
3: $tcpCongestionAvoidance$
4: **if** $(timeout)$ **then**
5: $tcpSlowStart$
6: **end if**

described as congestion-induced issues, thus activating the congestion avoidance mechanism, and reducing throughput. There exists substantial research into the performance of TCP in mobile networks (i.e., cellular communications); for example TCP variants such as CUBIC, Scalable and BIC has proven to achieve substantial throughput and fast window convergence in Long-Term Evolution (LTE) networks (4).

In this paper, we investigate a number of TCP congestion avoidance mechanisms implemented by the corresponding protocol variants, in order to deduce a minimum-compromise approach to achieve reliable and near real-time ad hoc connectivity in FANETs. This paper's results and simulation outcomes can be easily extended to other ad hoc network typologies, mobile or static deployments. Moreover, in this paper, we propose and investigate a new TCP variant, based on our comparative analysis and the extracted performance metrics and guidelines. Our proposed TCP variant aims to tackle transport-layer issues predominately present in wireless aerial ad hoc networks, with a focus on the examined drone swarm scenario.

The remainder of this paper is structured as follows. Section 2 is dedicated to the analysis of the twelve TCP variants which we considered for our evaluation, and the explanation of the core elements of the respective congestion control algorithms. Continuing, Section 3 engages in the actual comparative evaluation of the aforementioned congestion control algorithms. Statistical metrics are extracted, and various network and service-level attributes are interpolated to extract the maximum possible information. Section 4 proposes a new TCP variant, considering the findings of the comparative analysis. Section 5 discusses the results obtained during our experiments, and provides additional insight on the achieved data exchange. Lastly, Section 6 concludes the paper with some final remarks regarding future research.

2. TCP Variants

This section is dedicated to the high-level analysis of the TCP variants under investigation. In this work, we consider a spectrum of TCP variants, enumerated below. We computationally benchmark all below-mentioned and perform a detailed comparative analysis, which is correspondingly documented in Section 3. We analyse and benchmark New Reno, Hybla, High Speed, HTCP, Vegas, Scalable, Veno, BIC, YeAH, CUBIC, Westwood Plus, and Illinois.

A total of twelve TCP variants are analyzed and eventually benchmarked in a high-mobility lossy environment, introduced by inter-UAV communications in FANETs. Particular emphasis is placed on attributes affecting or directly affected by high node mobility and lossy channels. The high-level descriptions and theoretically expected behaviours are compared side-to-side with the results actually obtained during the NS3-based simulations. Advantages and shortcomings of the TCP variants examined are highlighted, with the ultimate goal of eliciting the most optimal congestion control mechanism for high-speed and lossy ad hoc networks.

2.1. TCP Reno

TCP (New) Reno can detect multiple packet losses, and this allows the network to come out of the congestion state easily. It is thus effective in terms of handling packet loss in volatile environments, and As this event is quite common in wireless ad hoc networks (the network typology of interest for this work), this variant can be considered as a solid performance benchmark. TCP Reno implements fast-retransmit upon reception of multiple duplicate packets and remains in fast-recovery mode until all transmitted data has been successfully acknowledged (ACK'ed). One more key factor differentiating Reno from other variants is the fact that the fast-recovery phase allows for multiple packet re-transmissions. In the fast-recovery phase, when a new acknowledgement (ACK) is received, there are two options: in the case of a full ACK, then Reno exits fast recovery and sets the cwnd equal to the slow start threshold. It then continues implementing congestion avoidance. However, in the case of a partial ACK, Reno considers that the next segment in line has been lost; it thus retransmits the affected segment and sets the number of duplicate acknowledgements received to 0. It subsequently exits fast recovery, when all the data is successfully acknowledged. One weakness of New Reno is that it take one complete RTT cycle to detect each packet loss. Therefore, it reacts relatively slowly in the event of multiple packet losses.

2.2. TCP Hybla

TCP Hybla was initially proposed to solve problem of high RTT in satellite links. The purpose of Hybla is to achieve approximately the same data transmission rate as that of a reference wired connection (5). Considering that this variant is designed to work with long-distance aerial links, it is considered appropriate to investigate it in our present work, associated with UAV networks. Hybla attempts to target improvements based on the analytical evaluation of the channel's congestion status. Interestingly, Hybla also attempts to draw conclusions using a metric derived from the frequency with which the congestion window changes. The congestion control algorithm of TCP Hybla will result in a greater average congestion window. However, this congestion control algorithm introduces the possibility for multiple losses in the same window, especially in long RTT connections (5), this effect is important to note, but given that our networks of interest are composed primarily of UAVs, the RTT will not be comparable to that of satellite links. Nevertheless, this phenomenon may result in the slowing down of the average throughput, as the transmission rate drops while activating the congestion avoidance process.

2.3. TCP High Speed

The High Speed congestion control algorithm for the TCP protocol has been defined in RFC3649 (6) and was introduced by S. Floyd et al. (7). The motivation behind this development is that conventional TCP performs rather poorly in networks with a large Bandwidth-Delay Product (BDP), which can be defined as "the product of a link's capacity (bits per second) and the round-trip delay time (seconds)": $B \times D = \frac{bits}{seconds} \cdot seconds$. The bandwidth-delay product is measured in bits and corresponds to the data that has been transmitted, but not yet acknowledged, which is maximum amount of data on the link at any given time. Networks with a large bandwidth-delay product are called Long Fat Networks (LFN), with the threshold for a classification being 10^5 bits. While TCP Reno is unable to fully utilize available bandwidth in such networks, this variant makes some minor yet essential modifications to the standard congestion control mechanism to overcome this limitation. When the congestion window is small, High Speed TCP behaves as standard TCP would. However, When TCP's congestion window is beyond a certain threshold, its behaviour changes, in the scope of increasing the window at a greater rate, and also recovering from potential losses more quickly. Thus, this variant of TCP can easily utilize bandwidth in LFNs to a greater extend. This TCP variant was

developed to sustain high speeds without the requirement for unrealistically low loss rates, while reaching said high speeds reasonably quickly when in slow-start mode. Lastly, High Speed can recover from multiple time-outs or other periods with small congestion windows without disrupting delays.

2.4. HTCP

HTCP is a TCP variant introduced by D. Leith and R. Shorten (8). It was also developed to tackle networking issues in LFNs. Similarly to High Speed TCP, HTCP is a loss-based algorithm. This particular TCP variant is geared towards increasing the aggressiveness of congestion window additive increase, so as to secure appropriate communication capacity in LFNs, all whilst maintaining TCP friendliness for small BDP links. This TCP variant increases its aggressiveness as the time since the previous loss increases. This method is finds particular applicability at the rate of additive increase, and not so much at the rate of multiplicative decrease. This more aggressive approach enables HTCP to avoid the issue encountered by other variants (e.g., High Speed TCP and TCP Bic) associated with making flows more aggressive if their windows are already large, which in turn can lead to congestion in LFNs. In this manner, HTCP can achieve convergence to fairness faster than its predecessor, High Speed TCP or TCP Bic. Nevertheless, this variant is not without its drawbacks: it is possible that due to this more aggressive approach towards additive increase, bandwidth abuse can occur on behalf of flows without observed packet losses. Essentially, lack of packet losses on a flow will lead HTCP to increment its congestion window in an unfair manner, thus reducing overall throughput. Overall, this variant appears to be able to offer greater throughput, but not necessarily a greater congestion window, due to the manner in which it handles the existence of multiple flows of varying packet drop rates.

2.5. TCP Vegas

TCP Vegas is a congestion avoidance algorithm that places particular emphasis on packet delay, rather than packet losses, as a metric to modify the congestion window and compute the rate at which to transmit packets. It was initially developed by Lawrence Brakmo and Larry L. Peterson (9) and is designed to detect congestion events at an emergent stage depending on determining increasing Round-Trip Time (RTT) values of packets in the established link. This is in stark contrast to other TCP variants (e.g., New Reno) that can only detect a congestion event after it has occurred, through an observed packet loss. Consequently, this specific TCP variants is heavily dependent on accurate calculation of the base RTT of a flow, and its corresponding value fluctuation: should the base RTT be too small, the throughput of the connection will be measurably less than the available bandwidth, and if the value is too large, the throughput will overrun the connection. It can be deducted that in highly mobile links (e.g., in FANETs), where RTT is low, and the duration of the established links is relatively small, throughput will be significantly affected. However, this is heavily dependent on the mobility model assumed by the networked nodes. In a coordinated mobility model, such as the Gauss-Markov mobility model, link duration may improve, but small RTT values of packets may still negatively affect throughput. Nevertheless, this TCP variant promises low end-to-end delay, resulting in measurably more responsive communications.

2.6. Scalable TCP

Scalable TCP was introduced by R. Morris in 2000 (10). It differentiates itself from TCP New Reno by introducing a new algorithm for modifying the congestion window. Instead of reducing the congestion window size by 50% (which is the case in TCP New Reno), each respective packet loss event results in the decrease of the congestion window by a factor of 12.5%, until no further packet loss is observed. When no more packet losses occur, packet transmission rate is increased at a fixed rate and the congestion window also increments in an additive manner. More specifically, one packet is added to the next transmission queue, for every hundred successful ACKs, instead of the rate observed by TCP New Reno, which is equivalent to the inverse of the size of the congestion window at a the instance where the fast recovery mechanism is activated. Theoretically, this will help alleviate the issue of large congestion windows taking too long to recover from congestion events, which is standard in classical TCP. This variant is supposed to increase throughput in channels with high loss rates, and targets the additional issue observed in many networks, where long and non-deterministic recovery delays are caused by the fact that transmitting nodes often use conservative loss detection and re-transmission mechanisms. It can be assumed that this TCP variant will provide a better average congestion window in high-loss rate channels, such as those corresponding to FANETs. However, the above-described congestion avoidance mechanism may also introduce additional, yet deterministic, end-to-end delay.

2.7. TCP Veno

TCP Veno was introduced by C. P. Fu and S. Liew (11). This TCP variant is envisaged to enhance the Reno congestion avoidance algorithm in terms of handling and recovering from random packet loss in wireless networks. This enhancement is implemented by estimating backlog of the transmission queue to be able to distinguish between nominal and congestive flows. This mechanism is envisaged to address the emergence of network heterogeneity, in which different network segments may have widely different link-layer attributes. In terms of manipulating the congestion window, TCP Veno maintains the core idea of TCP Reno, meaning that the size of the congestion window is increased progressively when no packet loss is observed. TCP Veno employs the original slow-start algorithm as TCP Reno as-is, without any modifications. It however tampers with the additive increase algorithm by making the following change: if the backlog packets exceed those in the buffer, the algorithm will increase the window size by one - this will happen every two RTTs, so that the connection can maintain this operation for as long as possible and thus achieve stability. TCP Veno also modifies the multiplicative decrease component of classical TCP. In the case where TCP Veno is in a non-congestive state and a packet loss is encountered, it will decrease its cwnd by 20%. The reasoning behind this implementation choice is that the encountered packet loss encountered is more likely corruption-based rather than congestion-based. However, if the backlog is indeed greater than the queue anda such event is observed, TCP Veno will decrease the congestion window by 50%, as is the case in Reno. This approach is similar to the one followed by the above-described TCP Scalable.

2.8. BIC TCP

The Binary Increase Congestion control (BIC) algorithm was first introduced by L. Xu et al. (12). As is the case with other examined congestion control algorithms, this variant aims to address issues observed in LFNs. More specifically, delays observed in such networks constitute a unique environment which can potentially cause issues to classical TCP in terms of fully utilizing available bandwidth. Other protocols consider two pivotal properties, namely TCP friendliness and bandwidth scalability: a protocol should not abuse available bandwidth on behalf of distinct flows, but should also not leave channel bandwidth idle in high-speed networks. This variant targets the aforementioned challenges, as well as another one: RTT unfairness in the case of competing flows. The reason behind flows with varying RTTs consuming unfair portions of bandwidth, is that the congestion window increase rate is enlarged as the window itself grows. This congestion control mechanism utilizes two window size control policies, namely additive increase and binary search increase. In the case of a congestion event, TCP BIC takes as a starting point the current congestion window value, and as a target point the value just before the loss event. At that point, the protocol uses a binary search technique to update the congestion window value at the midpoint between the two; this can be done either directly or by using an additive increase strategy, in case the distance between the current and the target window is too large. Thus, the congestion window can approach its maximum value in a logarithmic fashion, until the difference between these two values falls below a defined threshold. This approach can promise an increased average congestion window, at the cost of a potentially higher average end-to-end delay.

2.9. CUBIC TCP

TCP CUBIC was proposed by S. Ha et al. in 2008 (13). Its name is derived from the fact that the congestion window is a cubic function of the time elapsed since the latest congestion event, instead of a linear function of said metric. The exponential nature of the evolution of the congestion window entails a better-stabilized congestion window region which is defined as the "plateau" between the concave and the convex portions of the maximum congestion window over time. Given the fact that in almost all links, TCP CUBIC spends a measurable portion of its duty cycle in this plateau, it is capable of optimizing bandwidth utilization for the given channel, even more so in the face of high latency in LFNs. Another key differentiating factor between CUBIC and other TCP variants is that CUBIC does not rely on the observed pattern of RTTs to appropriately adjust the congestion window size, as it is only dependent on the last congestion event. This enables CUBIC to be more fair than other variants, which is not the case with other variants where ACKs are received faster and thus have their corresponding windows growing at a faster rate than other flows with longer RTTs. This functionality of TCP CUBIC may prove to be invaluable for high-mobility deployments, given the potential variance of RTTs and overall instability of established ad hoc links.

2.10. YeAH TCP

Yet Another HighSpeed TCP (TCP YeAH) was proposed by A. Baiocchi et al. in 2007 (14). It is a heuristic variant of TCP, and was initially designed to balance several varying requirements, such as fully exploiting the link

capacity of LFNs, inducing the smallest possible number of congestion events, being compatible with other TCP flows (predominately Reno), all while being capable to achieve RTT fairness and demonstrating robustness against random losses. Lastly, TCP YeAH is envisaged to achieve sufficiently high performance regardless of buffer size. TCP YeAH updates the congestion window via two operation modes. The congestion control algorithm switches between a "slow" mode state, during which the congestion window is nominally updated using the fundamental Reno algorithm, and a "fast" mode state, during which the congestion window is being updated using the High Speed TCP. It can be argued that TCP YeAH is not as much of a distinct congestion control protocol, as it is an algorithm for switching between said "slow" and "fast" states. The aforementioned switching process is a direct function of the number of queued packets in the bottleneck buffer. It should also be noted that this TCP variant employs TCP Vegas' mechanism for calculating the aforementioned backlog. After receiving three duplicate ACKs, and thus entering fast recovery mode, TCP YeAH decreases its slow start threshold as follows: $ssthresh = min(max(cwnd/8, Q), cwnd/2)$, where $Q = (RTT - BaseRTT) \cdot (cwnd/RTT)$. This method is implemented under the assumption that the TCP YeAH flows are not competing with TCP Reno flows - should that be the case, the slow start threshold is reduced by 50%, as in Reno-based variants.

2.11. TCP Westwood

TCP Westwood Plus was first implemented by A. Dell'Aera et al. in 2004 (15). It constitutes an evolution of TCP Westwood which was initially introduced by S. Mascolo et al. in 2001 (16). Essentially, it is a transmitter-side modification of TCP Reno. Its main purpose is optimizing congestion control via end-to-end bandwidth estimation. By estimating bandwidth, Westwood Plus can efficiently set the congestion window and the slow start threshold, after experiencing a congestion event which is indicated after three duplicate ACKs or a timeout. TCP Westwood considers the current values of the congestion window, the slow start threshold, the established connection's RTT, as well as the minimum round trip time which is measured by the transmitting node. The fundamental idea behind TCP Westwood is using the stream of returning ACK packets in order to estimate the available bandwidth of a connection. As mentioned above, this estimate is used to properly set the cwnd and ssthtresh when a congestion event is detected. If no congestion event is detected, those variables default to conforming to RFC2581 (17). The new bandwidth estimation algorithm used by Westwood Plus relies on bandwidth samples collected every single RTT, instead of every time the transmitting node receives an ACK. Essentially, the bandwidth estimate at a given time n can be defined as the output of the following exponential filter: $BWE_n = \frac{7}{8} BWE_{n-1} + \frac{1}{8} B_n$, where B_n is a bandwidth sample. Bandwidth samples are calculated as follows: $B_n = D_n/T_n$, where D_n is the amount of data in bits, and T_n is equal to the value of RTT in seconds at a given instance. This method is envisaged to enable TCP Westwood Plus to achieve greater throughput, without much added computational overhead.

2.12. TCP Illinois

TCP Illinois was first proposed by S. Liu et al. in 2006 (18). Similarly to TCP Westwood and Westwood Plus, this variant is essentially a transmitter-side modification of the standard TCP congestion control mechanism. TCP Illinois focuses on high-speed, long-distance networks and is capable of achieving a higher average throughput than TCP Reno, by implementing more fair and effective network resource utilization, in a manner that enables compatibility with nominal TCP Reno. This variant uses information on packet loss as a means of determining the appropriate size of the congestion window, while utilizing queuing delay information to compute the necessary increment size for the congestion window to either increase or decrease. Overall, TCP Illinois manages to increase throughput more quickly than TCP Reno while no congestion event is detected, and increases throughput conservatively when a congestion event is imminent. Consequently, obtained throughput is higher than that of TCP Reno, while maintaining responsiveness. Moreover, given that this TCP variant is aware of packet loss (as this information is utilized as a transmission handling metric), it is possible that TCP Illinois will behave well in high-mobility environments, where links do not demonstrate reliability and packet loss is rather frequent (18).

3. Comparative Evaluation

This section is dedicated to the comparative analysis of the identified TCP variants. We developed an NS-3-based framework to simulate point-to-point communication between networked nodes. We utilized the Gauss Markov Mobility Model, with mean velocity being randomly defined, ranging between two predefined thresholds. Table 1 compares the involved TCP variants by using a set of metrics such as received packets, received bytes, throughput,

Table 1
Comparison of flow statistics of examined TCP variants

	TCP Variants											
Flow Statistics	**New Reno**	**Hybla**	**High-Speed**	**HTCP**	**Vegas**	**Scalable**	**Veno**	**BIC**	**YeAH**	**CUBIC**	**Westwood Plus**	**Illinois**
Tx Packets (No)	96632	97510	97563	97498	97438	96967	96632	96238	96624	97440	97284	97498
Rx Packets (No)	96519	97363	97364	97363	97360	96676	96521	95951	96517	97361	97169	97364
Delivery Ratio (%)	99.88	99.85	99.80	99.86	99.92	99.70	99.86	99.70	99.89	99.92	99.88	99.86

Table 2
Simulation Parameters

Attribute	**Value**
Number of Nodes	10
Position Allocator	Random Rectangular Position
Mobility Model	Gauss-Markov Mobility Model
MAC Standard	802.11b
Packet size	536 bytes
Transmission Queue Class	FIFO Drop Tail Queue
Simulation Time	100 s
Peak Node Velocity	20 m/s
Node Pause Time	0.1 s
TCP Variant	New Reno, Hybla, High Speed, HTCP, Vegas, Scalable, Veno, BIC, YeAH, CUBIC, Westwood Plus, Illinois

mean delay, mean jitter, and packet delivery ratio (PDR), given a set of input parameters, namely transmitted packets, transmitted bytes, and offered load.

3.1. Simulation Scenario

The total number of nodes has been set to 10, and the number of sinks to 10. To introduce some challenge in the establishment and maintenance of network links, mean node velocity is assumed to be equal to 20 m/s (72 km/h). This high speed will allow us to benchmark the capability of each TCP variant to maintain an active communication channel in such volatile environments. Consequently, the unique characteristics of each variant and the way they handle congestion will be significantly highlighted, and more valuable results will be obtainable. Figure 1 showcases the approximate starting positions of the 20 nodes participating in the routing experiment. Table 2 summarizes the parameters of the NS3 simulation we designed. We defined the total number of nodes to be equal to ten.

We utilized the random rectangular position allocator for the distribution of the nodes inside a 3D grid. The MAC standard we used is the 802.11b, which supports 140 meters of outdoors range, and transfer rates of up to 11 Mb/s. Moreover, we assumed a 5Mb/s channel data-rate and a First-In-First-Out drop tail queue for the creation of data packets to be transmitted. Simulation time was limited to 100 seconds, and node mobility was set to 20 m/s, with a pause time equal to 0.1 seconds. For the mobility of the networked nodes, we used the Gauss-Markov mobility model. Mobility vectors for each node are therefore defined using two parameters: one for the new speed of a node (i.e., the velocity with which it will approach the next waypoint, after a pause time which we set to 0.1 seconds), and one for the new direction (i.e., the aforementioned waypoint) using a randomness index, introduced by random variables of Gaussian distribution. Equation 1 describes the process of computing the new speed of each node in the allocated 3D space at n^{th}. Similarly, Equation 2 describes the process for the definition of new direction/waypoint for new swarm nodes. Last but not least, Equation 3 describes the process for calculating the next pitch value of each airborne node. This parameter is valuable in our case, as it will be pivotal in computing the linear velocity vector to give to the respective NS3 helper module. In all Equations 1, 2 and 3, a indicates a randomness index outputted by a Gaussian Distribution.

$$s_n = as_{(n-1)} + (1-a)\bar{s} + \sqrt{(1-a^2)}s_{x_{n-1}} \tag{1}$$

$$d_n = ad_{(n-1)} + (1-a)\bar{d} + \sqrt{(1-a^2)}d_{x_{n-1}} \tag{2}$$

$$p_n = ap_{(n-1)} + (1-a)\bar{p} + \sqrt{(1-a^2)}p_{x_{n-1}} \tag{3}$$

Using the above-described parameters, the Gauss-Markov mobility model will eventually compute the actual mobility vector of each swarm node. More specifically, from Equations 1, 2 and 3 we can calculate the velocity in the 3D grid showcased in Figure 1 in each axis. Equation 4 describes the velocity, analyzed in each axis of the 3D grid.

$$velocity(x,y,z) = \begin{cases} s_n cos(d_n)cos(p_n), \text{x axis} \\ s_n sin(d_n)cos(p_n), \text{y axis} \\ s_n sin(p_n), \text{z axis} \end{cases} \tag{4}$$

$$\begin{aligned} \vec{V} &= (s_n cos(d_n)cos(p_n))\hat{i} \\ &+ (s_n sin(d_n)cos(p_n))\hat{j} \\ &+ (s_n sin(p_n))\hat{k} \end{aligned} \tag{5}$$

To eventually compute the vector for the next step, we follow the process described in Equation 5. Similarly, we can calculate the length of the vector and get thus obtain the final position of the node using the fundamental notion that $|V| = \sqrt{x^2 + y^2 + z^2}$. This value is directly sent to the NS3 mobility helper, which assigns it to the respective node. This procedure is implemented 10 times per unit of time (once per each node). It becomes clear that this mobility model uses the last speed and direction of each node, to determine the next ones by introducing some degree of randomness with each iteration. By modifying the above-described equations, we can introduce new, potentially more or less random mobility vectors to the swarm's behaviour.

Carefully checking the results documented in Table 1, we can deduct several conclusions. It is worth mentioning that the number of transmitted packets is not the same in all TCP variants, as we are using a First In First Out (FIFO) drop tail queue to formulate the packets to be sent. This implementation gives us further insight as to how each respective variant handles its offered load and can form the next packet. Essentially, we let each TCP variant transmit the maximum possible amount of data within a specified time-frame, each processing the offered load at their own respective rate, instead of forcing the same amount of packets through each TCP session, as the latter would mean that "faster" TCP variants would stop transmitting faster than "slower" variants (i.e., we would have implemented an event-driven simulation, whereas we are trying to emulate a time-driven simulation, emulating a total of 100 seconds streaming communication in a highly mobile environment). Consequently, as we do not want packet queues to become emptied and thus having a TCP variant stop transmitting while another one keeps its communication alive, we implemented the aforementioned FIFO drop tail queue.

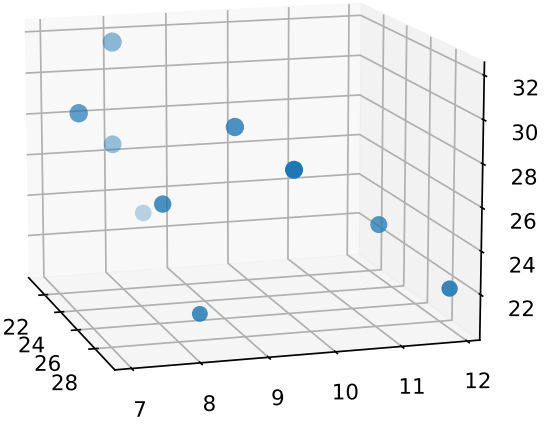
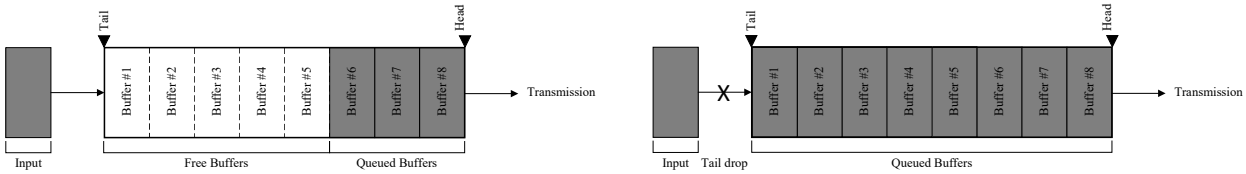


Figure 1: Initial positions of nodes in the 3D space

Figure 2a demonstrates how an incoming packet is normally handled by the drop-tail FIFO queue responsible for buffering incoming packets for transmission. Respectively, Figure 2b showcases how the same queue will behave if no more free buffers exist within said queue, in which case a tail drop event will occur.

We can deduct that in terms of processed load, TCP Hybla performs the best, and the TCP BIC performs the worst. This means that Hybla is capable of quickly iterating through the offered queue, but without necessarily ensuring reliable reception for all transmitted packets. Correspondingly, BIC does not seem to be as competent in iterating through the provided packet queue, but may prove to be better in deducing the optimal congestion window and thus achieving comparable throughput. As we can observe, this metric by itself does not provide enough data to judge the capacity of each congestion control algorithm. This is why we used the number of received packets to calculate the actual obtained PDR.



(a) Nominal FIFO queuing of packets, assuming the existence of free buffers in the transmission queue.

(b) Tail drop event in FIFO queue, assuming a lack of free buffers to hold the incoming packet.

Figure 2: Nominal FIFO queuing and drop tail event whilst buffering packets for transmission.

3.2. Throughput Evaluation

This subsection analyses the examined TCP variants in terms of their achieved throughput and PDR. TCP Reno seems to be a good, all-around variant. It boasts a high PDR (99.88%), and a sufficient balance between offered load and throughput. The comparatively high PDR is a direct consequence of TCP New Reno's capability of detecting multiple packet losses - it is thus much more efficient that in the event of multiple packet losses. It is worth mentioning that the main difference between the original version of TCP Reno and TCP New Reno, is that the former can't distinguish between full ACK and partial ACK, while the latter can. As such, multiple packet loss is detected and can be mitigated by New Reno. TCP New Reno remains in the fast recovery until all the outstanding packets are properly acknowledged. This gives this variant a head-start in terms of reliability, but also a handicap in terms of scalability and responsiveness. Nevertheless, considering the overall performance of TCP New Reno in this first evaluation, we will utilize it as a performance baseline for evaluating the rest of the identified variants. The two main metrics of interest is congestion management (to evaluate responsiveness and scalability), and reliability (to evaluate achievable Quality of Service (QoS)).

Westwood Plus performed quite similarly to TCP Reno, achieving the same PDR. Nevertheless, Westwood Plus managed to transmit and acknowledge an additional 650 packets, translating to 348400 extra bytes in comparison to Reno.

Continuing, TCP Hybla achieved a similar, yet slightly inferior PDR of 99.85%. TCP High Speed achieved a PDR of 99.8%. HTCP managed to obtain an increased PDR of 99.86% - same as TCP Veno and TCP Illinois (with an accuracy of two decimal points). However, Illinois managed to transmit and acknowledge an additional 1 packet, translating to 536 extra bytes, in comparison to HTCP, which in turn transmitted an additional 451312 bytes, translating to 842 extra packets during the same time-frame.

Impressively, TCP Vegas achieved the highest observed PDR of all variants, equal to 99.92% - the same as CUBIC. Nevertheless, TCP CUBIC managed to transmit and acknowledge an additional 1 packet , translating to 536 extra bytes in comparison to Vegas. Therefore, the performance of Vegas and CUBIC is almost identical in terms of acknowledged packets. Regarding TCP Vegas, its outstanding performance on a per-byte basis can be attributed to the fact that this variant is designed to detect congestion events at an emergent stage, instead of post-occurrence, by determining the rate at which the RTT increases. As for CUBIC, its overall stability (explained through the existence of the plateau between the concave and convex portions of the congestion window) seems to be the core attribute enabling an increased performance in terms of PDR.

TCP Scalable proved to be the least reliable in terms of packet delivery, achieving a PDR of 99.7%, which is the same value as that of BIC. It should be noted however, that this PDR value is certainly not inadequate, as it goes well beyond 95%. Again, the core difference between Scalable and BIC is that the former actually managed to successfully transmit and acknowledge an additional 725 packets, translating to an additional 388600 bytes of data. TCP YeAH achieved a PDR of 99.89% in the 100 seconds of the simulation.

Continuing, Figure 3 expresses the throughput obtained by each respective TCP variant, on a per-second basis. We can observe that again, almost all TCP variants follow a similar trend. Initially, most TCP variants demonstrate a sudden drop of throughput. TCP BIC has the greatest drop by comparison, reaching approximately 230000 bytes per second. TCP Scalable behaves very similarly, reaching 300000 bytes per second right before increasing to about 570000 bytes per second, and then fluctuating between 450000 and 550000 bytes per second.

Interestingly, TCP YeAH seems to have an abrupt drop of throughput around the 25 seconds mark, and stabilizes again within a few seconds. TCP Westwood Plus also behaves in a similar manner, but only appears to minimally drop

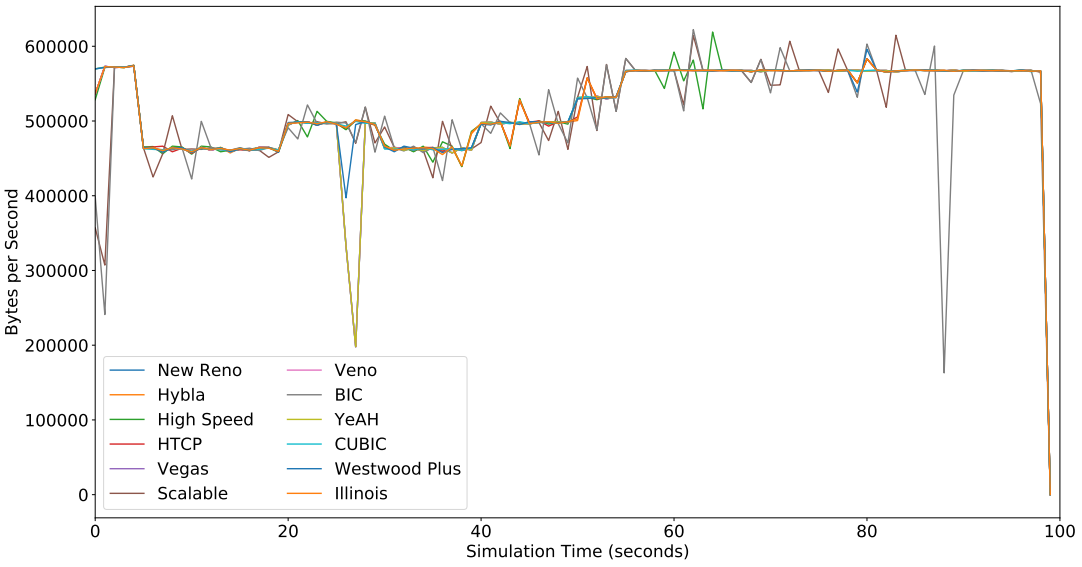


Figure 3: Throughput dynamics for the considered TCP variants

its throughput at the same timestamp. Last but not least, around the 90 seconds mark, TCP BIC has an equally abrupt drop of throughput. Finally, all variants are synchronized again, up until the point where the communication seizes. The most probable reason as to why some TCP variants demonstrate such abrupt throughput drops are congestion events. As a next step, we can examine more closely the obtained throughput for each TCP variant.

Figure 4 gives us further insight as to how each variant handles channel resources. All TCP variants managed to process load equal and greater than 90% of the channel data-rate. This is directly related to the way in which each variant handles the drop tail queue shown in Figures 2a and 2b. We define processed load as the amount of data iterated through the drop tail queue used to prepare packets to be sent, expressed as the percentage of available data-rate offered by the channel. Essentially, this metric expresses the efficacy of each variant in terms of utilizing the resources theoretically available to it.

Similarly, we define throughput as average rate at which packets are received and acknowledged, expressed as a percentage of the available channel data-rate. TCP New Reno performed below average, with an average throughput of 90.8016% of the available data-rate. TCP Hybla performed remarkably well in terms of resource utilization, with its throughput averaging at a 91.5974% of the available channel data-rate, same as HTCP. TCP High Speed achieved an average throughput of 91.5984% of the available data-rate, which is the same as that of TCP Illinois; this is the maximum obtained value achieved by all variants.

High-Speed TCP measures the congestion window, and when it exceeds a certain threshold, it changes its behaviour changes so as to increase the window faster and recover from losses in a timely manner, which seems to be the key attribute enabling this performance. As for Illinois, the utilization of packet loss data as a metric to modify the congestion window, accompanied with the consideration of queuing delay data to compute the optimal congestion window increment size seem to pivotal in supporting high average congestion windows.

TCP Vegas also performed better than average, with its throughput reaching to 91.5946% of the channel data-rate. TCP Scalable performed slightly worse, with an average throughput of 90.9492% of the channel data-rate. What is interesting in this case, is that the gap between the processed load and throughput is increasing to 0.2738%. This fact hints to the possibility that this variant suffers from high end-to-end delays. TCP Veno demonstrates a throughput of 90.8034% of the channel data-rate. Similarly to Scalable, TCP BIC also shows a disproportionate gap in processed load and throughput. Again, this suggests that those two variants are susceptible to higher average end-to-end delays in highly mobile networks with low link lifetime. TCP YeAH shows a comparatively increased performance reaching

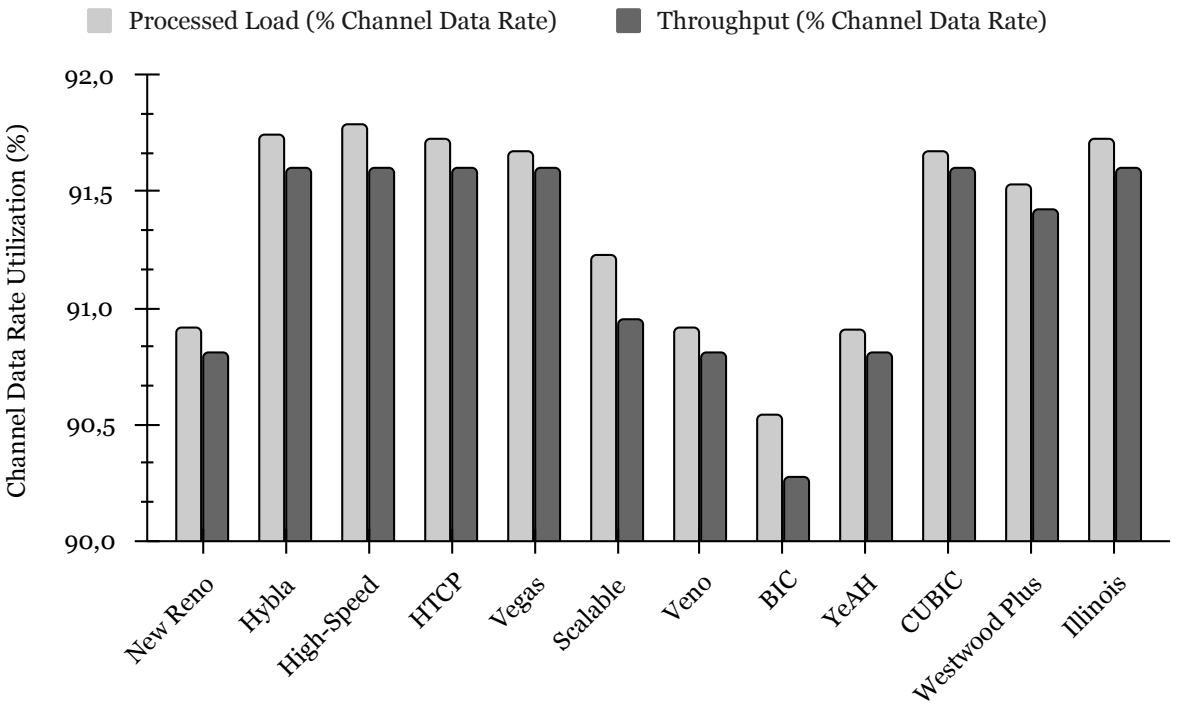


Figure 4: Processed load and throughput expressed in percentages of channel resource utilization (available data-rate percentage) for the considered TCP variants

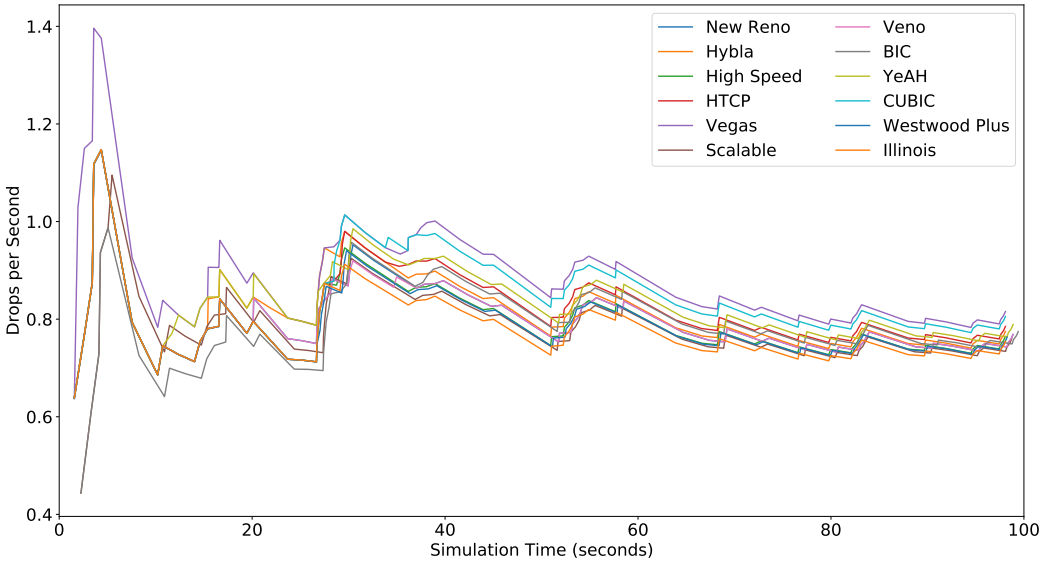


Figure 5: Observed packet drop events per second for the duration of the communication for each TCP variant

90.801% of the channel data-rate, while TCP CUBIC demonstrates a substantially increased performance of 91.5956% of the available channel resources. TCP Westwood Plus also performed well, managing a throughput of 91.4154% of the channel data-rate. In the case of TCP Scalable and BIC, the gap between the processed load and throughput is disproportionally increased.

Concluding our throughput analysis, Figure 5 is a visualization of the observed packet drops on a per-second basis. Right after the start of the communication, we observe a spike in packet drops for all TCP variants. TCP Vegas however, spikes above all other variants, peaking at 1.4 packet drops per second. The variant demonstrating the lowest spike in drops per second is TCP BIC. On a per-second basis, all variants seem to converge at 0.8 packet drops per second. After the thirty second mark, a clear pattern emerges: TCP Illinois steadily maintains the lowest number of packet drops per second, followed by TCP Scalable and TCP High Speed. Correspondingly, TCP Vegas appears to have the greatest number of packet drops per second, followed by TCP CUBIC and TCP YeAH.

3.3. Congestion Window Volatility Evaluation

Continuing, we used the congestion window data outputted by NS3 to compute the Cumulative Moving Average (CMA) of the absolute difference per each congestion window alteration. Equation 6 describes the process of calculating the CMA of the absolute difference of cwnd values at any given point n. We wrote a simple python script to compute and plot this statistical parameter. The output is demonstrated in Figure 6. The core idea is simple yet highly effective in visualizing the degree of variability of the congestion window for each TCP variant, throughout the course of the communication.

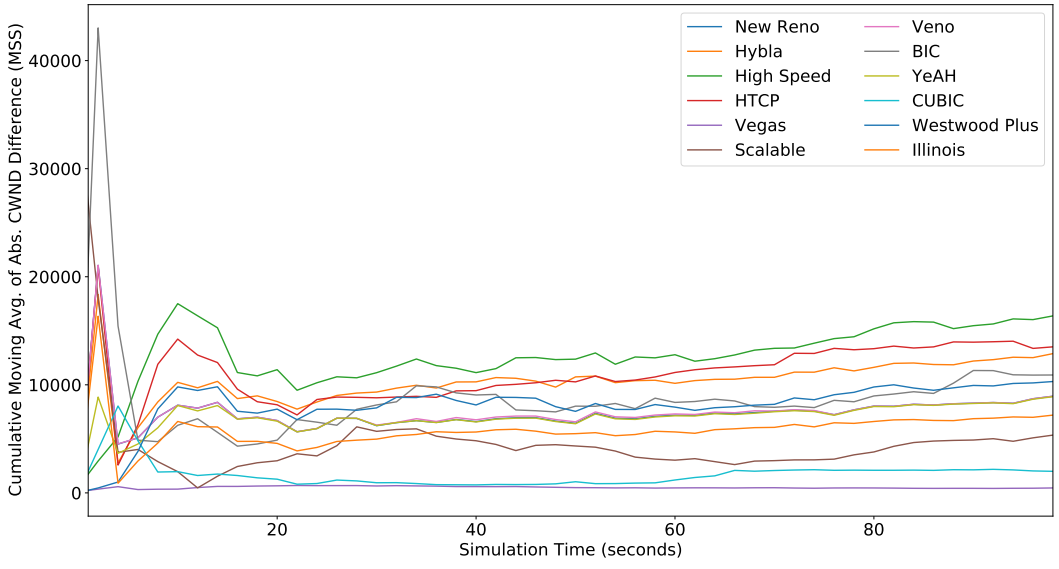


Figure 6: Cumulative moving average of absolute difference in congestion window for the considered TCP variants

By interpolating this data with the rest of the metrics which we compute and consider (i.e., throughput, packet drops, end-to-end delay, jitter, processed load etc), we can obtain greater insight into the core functionalities and behaviour of congestion control algorithms in mobile, three-dimensional aerial networks. Algorithm 4 describes in detail the process for obtaining the measurements showcased in the corresponding Figure 6.

$$\begin{aligned} CMA_n &= \frac{|cwnd_0 - cwnd_1| + ... + |cwnd_{n-1} - cwnd_n|}{n} \\ &= \frac{1}{n}\sum_{i=1}^{n} |cwnd_{i-1} - cwnd_i| \end{aligned} \tag{6}$$

Algorithm 4 Algorithm for calculating the cumulative rolling average of absolute differences of consecutive congestion windows

Input: $(cwnd_{n-1}, cwnd_n)$
Output: CMA

$i \leftarrow 0$
$j \leftarrow 0$
$y \leftarrow empty$
$CMA \leftarrow empty$
$rollSize \leftarrow 1$
for (every n) **do**
    $y[i] \leftarrow |(cwnd_{n-1} - cwnd_n)|$
    $i \leftarrow i + 1$
    while $(j < (length(y) - rollSize + 1))$ **do**
        $this\_RollSize = y[j : j + rollSize]$
        $avgRollSize = sum(this\_RollSize)/rollSize$
        $CMA.append(avgRollSize)$
        $j \leftarrow j + 1$
        $rollSize \leftarrow rollSize + 1$
    end while
end for

As mentioned before, the cumulative moving average of absolute difference in the congestion window is a metric demonstrating how "volatile" each respective TCP variant is. With that being said, we observe that TCP High Speed demonstrates the highest average such value. Similarly, TCP Vegas demonstrates the lowest variance in congestion window; interestingly this variant, had the highest observed PDR in Table 1, second only to TCP CUBIC, whose variance in congestion window is similarly very low. A noteworthy observation is that TCP BIC appears to have a spike in congestion window variance during the initial stages of the simulated communication, then drops to a minimal value, and then increases almost linearly. Another remark is that TCP Veno and TCP YeAH appear to perform almost identically in terms of variance, with only slight variations being present in the initial stages of the communication. The rest of the TCP variants seem to follow the same approximate trend, encompassing a heightened variance at the initial stages, with a stabilization and a subsequent near-linear increase throughout the duration of the communication.

3.4. Responsiveness Evaluation

The above-mentioned results suggest that these variants suffers from high end-to-end delays. The corresponding performance of TCP Scalable in terms of delay in Figures 7 and 8 validate this suspicion: while the average congestion window is the highest in both TCP Scalable and BIC, delay is also at its highest for those two specific variants.

Figures 8 and 9 provide insight to the responsiveness and the observed jitter of each TCP variant. TCP New Reno managed to keep mean end-to-end delay at 19.31 ms. TCP Hybla achieved a mean end-to-end delay of 27.99 ms, while TCP High Speed was less responsive, with mean end-to-end delay being equal to 36.49 ms. However, HTCP performed better, at 18.76 ms. TCP Vegas performed the best of all variants, with a mean end-to-end delay of 2.54 ms. TCP Scalable performed the worst of all variants in terms of responsiveness, at 50.09 ms end-to-end delay. TCP Veno performed closer to average, at 19.30 ms, while TCP BIC performed only slightly better than Scalable, at 46.90 ms. Continuing, TCP YeAH performed very close New Reno and Veno, at 19.32 ms. TCP CUBIC performed close to Vegas, with its mean end-to-end delay equal to 5.09 ms, the second best performance. TCP Westwood Plus performed somewhat better than average, at 16.55 ms. Lastly, TCP Illinois was moderately responsive, at 20.30 ms. In terms of jitter, performance is rather similar as that of the same variants in terms of delay. TCP Vegas outperformed every other TCP variant, with a mean jitter of 0.42 ms. The other extremely responsive TCP variant, CUBIC, had a jitter of 0.45 ms. The highest jitter which was recorded, belongs to TCP Westwood Plus, and is equal to 50 ms.

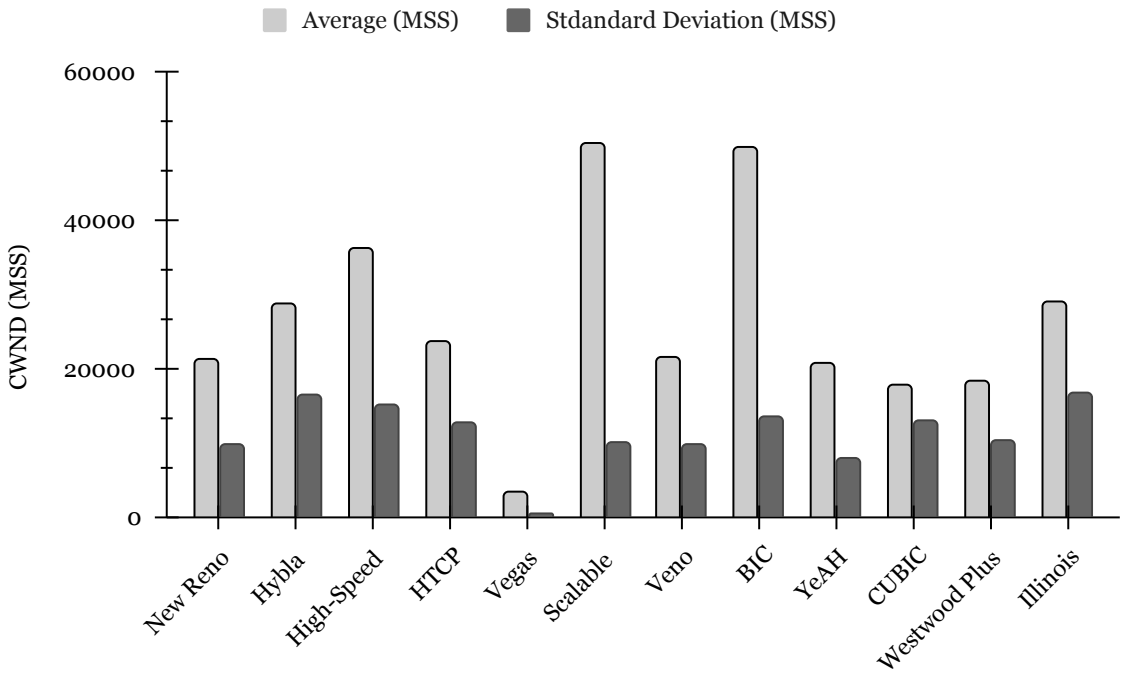


Figure 7: Average and standard deviation of the congestion window size for the considered TCP variants

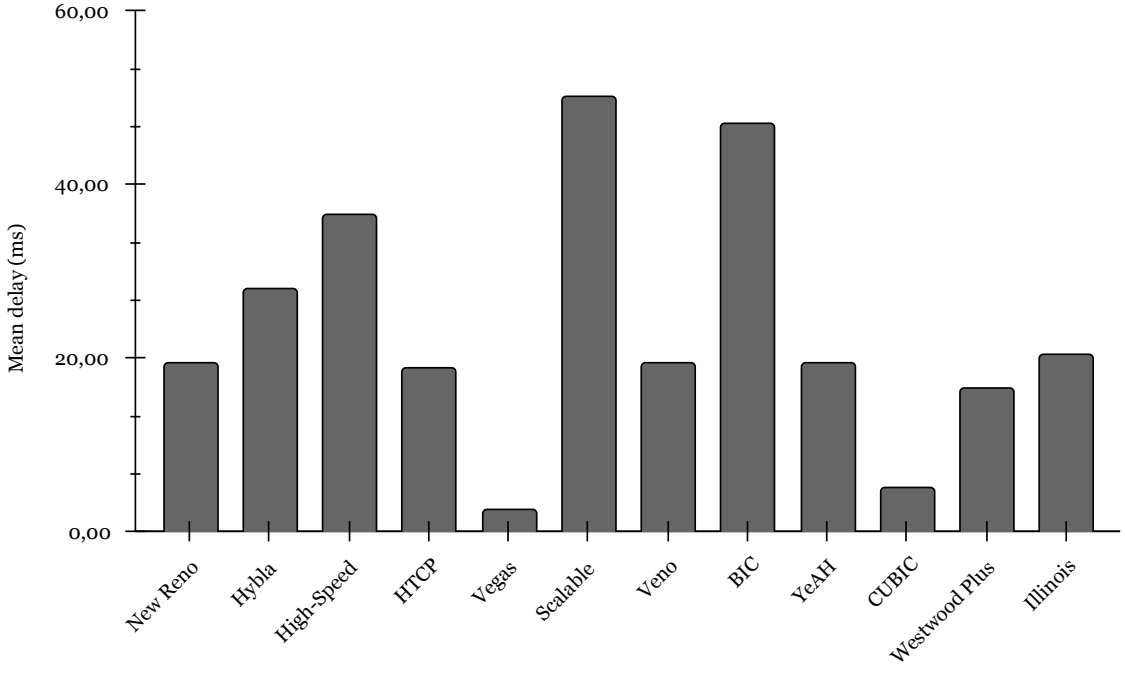


Figure 8: Mean end-to-end delay observed in the considered TCP variants

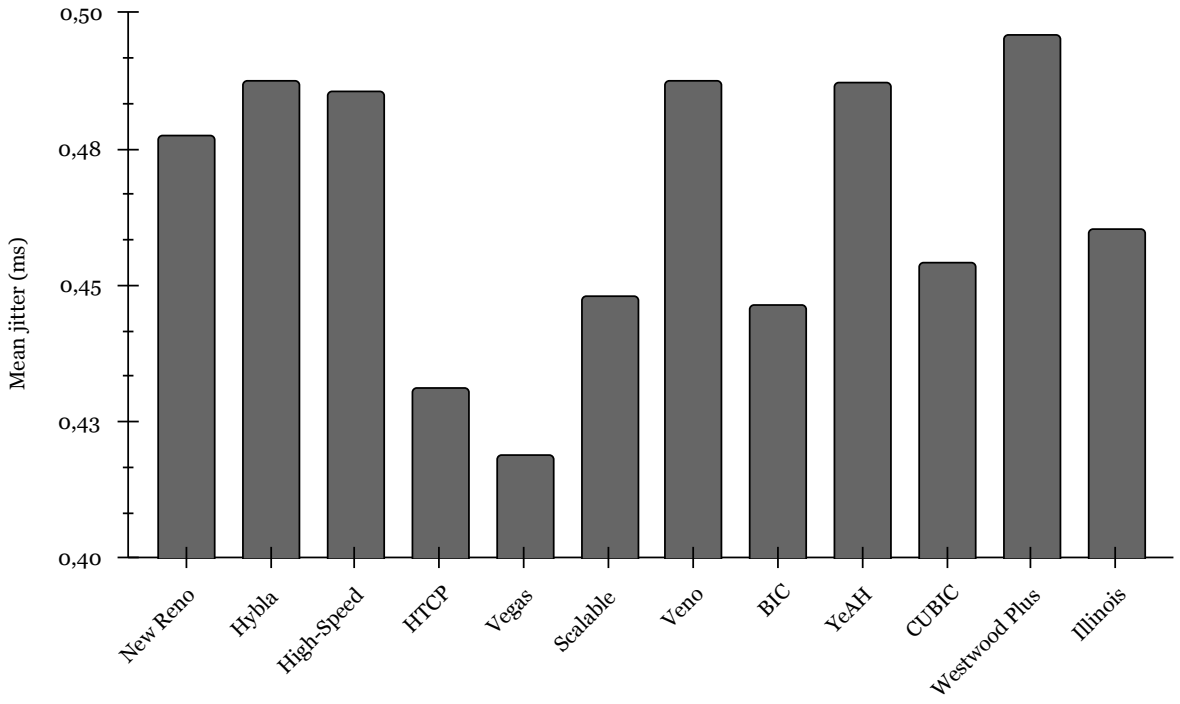


Figure 9: Mean jitter observed in the considered TCP variants

4. The Proposed S-HTCP Variant

From the analyzed TCP variants, HTCP seems to be the best-performing one, in terms of conjoint responsiveness and throughput. However, there seems to be room for improvement, in terms of end-to-end delay. For the discussed high-mobility UAV network scenario, the attribute whose increase needs to be directly addressed is overall responsiveness and throughput. As per D. J. Leith et al. (19), modifying the additive increase-multiplicative decrease (AIMD) algorithm can yield much better results in this regard. This is implemented by establishing a framework for adjusting the additive increase parameter "α" and the multiplicative decrease factor "β". In light of those remarks, we have identified a new methodology for dynamically adjusting the aforementioned parameters considering the time elapsed after the last observed congestion event, and the minimum RTT in a given flow. D. J. Leith et al. in (19) define the α factor as seen in Equation 7. The product is a loss- and delay-based algorithm capable of measurably increasing performance in ad hoc environments.

$$\alpha = \begin{cases} 1, \Delta \leq \Delta_L \\ 1 + 10(\Delta - \Delta_L) + \left(\frac{\Delta - \Delta_L}{2}\right)^2, \Delta > \Delta_L \end{cases} \tag{7}$$

Δ is the time in seconds since the last congestion event. Δ_L is the threshold for switching from standard/legacy operation to the new increase function, set to 1 second by default. Correspondingly, β is defined as seen in Equation 8.

$$\beta = \frac{RTT_{min}}{RTT_{max}} \tag{8}$$

At every successful acknowledgement, the additive increase factor gets updated: $\alpha \leftarrow 2(1-\beta)\alpha$, and the congestion window is adjusted: $cwnd \leftarrow cwnd + \alpha/cwnd$. Similarly, at every congestion event, β is re-computed according to Equation 8, and the congestion window is updated: $cwnd \leftarrow \beta \times cwnd$. The novelty behind the HTCP approach is that

it can adjust the additive increase rate to be greater for flows with a larger congestion windows. By scaling α with the RTT, HTCP renders the additive increase rate effectively invariant to RTT, along with convergence time. This approach limits RTT unfairness between competing flows.

We propose an AIMD scaling algorithm based on that of HTCP, modified for increased performance in high-mobility, aerial ad hoc networks: Swarm HTCP (S-HTCP). Our proposed approach involves enhancing the algorithm for the calculation of the α factor, by considering the time elapsed since the last congestion event and the current minimum RTT. This will enable the rate at which the congestion window increases to scale in an exponential manner. This makes the overall AIMD algorithm behave more aggressively my employing multiplicative incrementation of the congestion window depending on the time since the last observed congestion. Similarly, the behaviour of the β factor is modified to consider the time elapsed since the last congestion event, as a weighted metric. We define the two new AIMD factors for our derived variant as described by Equations 9 and 10.

$$\alpha_{S-HTCP} = \alpha e^{\Delta/\lambda_1 - RTT_{min}/\lambda_2} \tag{9}$$

The new α factor is described as a function of the classical HTCP one in Equation 9, while Equation 10 expresses the newly defined β_{S-HTCP} as a function of the original HTCP beta factor. We utilize a set of exponential decay functions to combine the Δ and RTT_{min} metrics into the α and β factors.

$$\beta_{S-HTCP} = \beta e^{-\Delta/\lambda_1} \tag{10}$$

Through experimentation, we deduced that the optimal exponential decay constants are: $\lambda_1 = 25$, and $\lambda_2 = 70$; these values were used throughout the duration of our testing. These values were carefully chosen, considering the rate at which congestion events occur in our aerial swarm scenario, and the (averaged) minimum RTT. Consequently, for our experimental research, the new α_{S-HTCP} factor can be expressed as a function of α as seen in Equation 11, while β_{S-HTCP} is expressed in Equation 10. Our experimentation in determining the optimal values for the exponential decay constants involved conjointly iterating the constants, until the desired effect was achieved for each AIMD parameter. The growth of the new α_{S-HTCP} factor is now appropriate and conjointly optimized to function with the newly implemented β_{S-HTCP} parameter.

$$\alpha_{S-HTCP} = \alpha e^{(14\Delta - 5RTT_{min})/350} \tag{11}$$

Considering the original definition of the α metric in Equation 7, and assuming TCP has already crossed the threshold for switching from standard operation to the new increase function, we can now express α_{S-HTCP} as in Equation 12.

$$\alpha_{S-HTCP} = e^{(14\Delta - 5RTT_{min})/350} \left(\frac{\Delta^2}{4} + 10\Delta + 1 \right) \tag{12}$$

Similarly, by taking the original definition of β of Equation 8 into account, we express β_{S-HTCP} as seen in Equation 13. It should be noted that in both cases (α_{S-HTCP} and β_{S-HTCP}) the time elapsed since the last congestion event (Δ) and the minimum RTT in a given flow (RTT_{min}) are the only unknowns at any given instance.

$$\beta_{S-HTCP} = e^{-\Delta/\lambda_1} \frac{RTT_{min}}{RTT_{max}} \tag{13}$$

This proposed approach builds on the fundamental HTCP algorithm, which inherently avoids scaling the AIMD factors as direct functions of the the congestion window, and allows TCP to demonstrate improved convergence after disturbances, which are rather frequent in aerial ad hoc networks, a common issue with BIC and High Speed TCP.

We proceed to compare the proposed TCP variant with its progenitor, HTCP. Figure 10 showcases the behaviour of the proposed TCP variant in terms of processed load and achieved throughput, expressed in terms of available bandwidth utilization. Evidently, S-HTCP significantly outperforms its counterpart, as it achieved a 3.8942% performance increase in throughput. This is a direct consequence of the utilization of the enhanced α and β metrics for HTCP's AIMD algorithm.

Figures 11a and 11b qualitatively showcases how S-HTCP improves responsiveness and significantly reduces jitter, effectively offering better services for delay-sensitive applications requiring reliable communications.

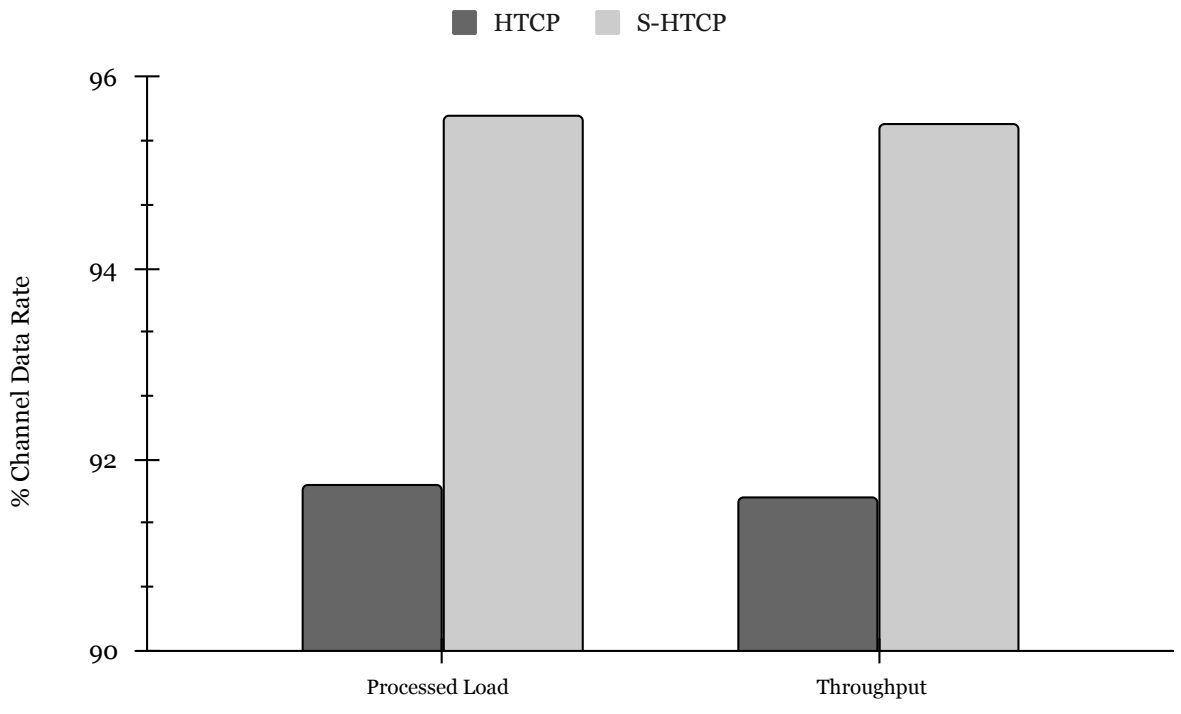
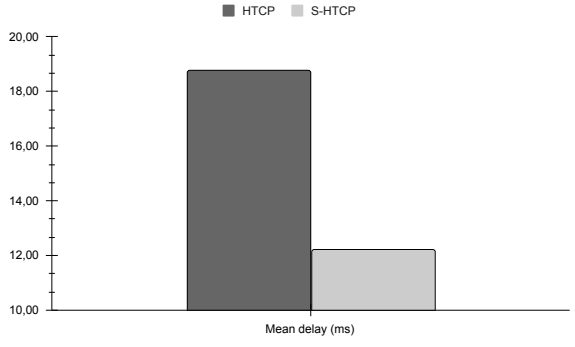
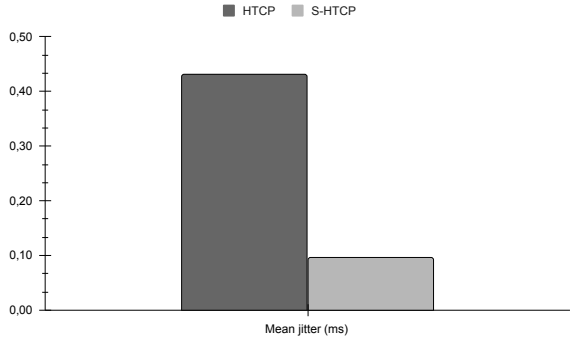


Figure 10: Processed load and throughput of HTCP and S-HTCP, expressed in percentages of channel resource utilization (available data-rate percentage)



(a) Comparison of mean end-to-end delay between S-HTCP and HTCP



(b) Comparison of mean jitter between S-HTCP and HTCP

Figure 11: A comparison of the mean end-to-end delay and Jitter observed in HTCP and S-HTCP

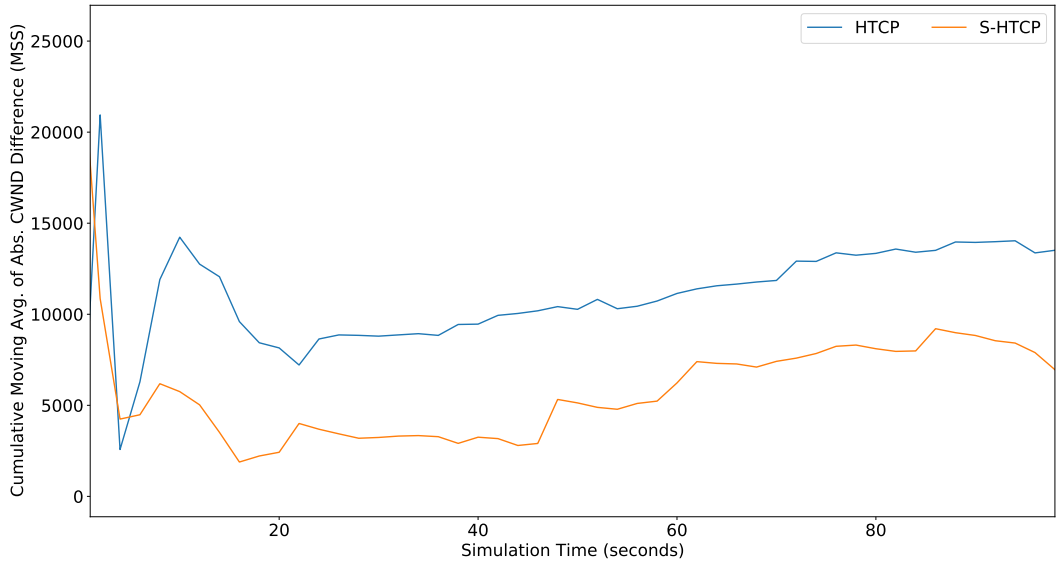


Figure 12: Cumulative moving average of absolute difference in congestion window of HTCP and S-HTCP

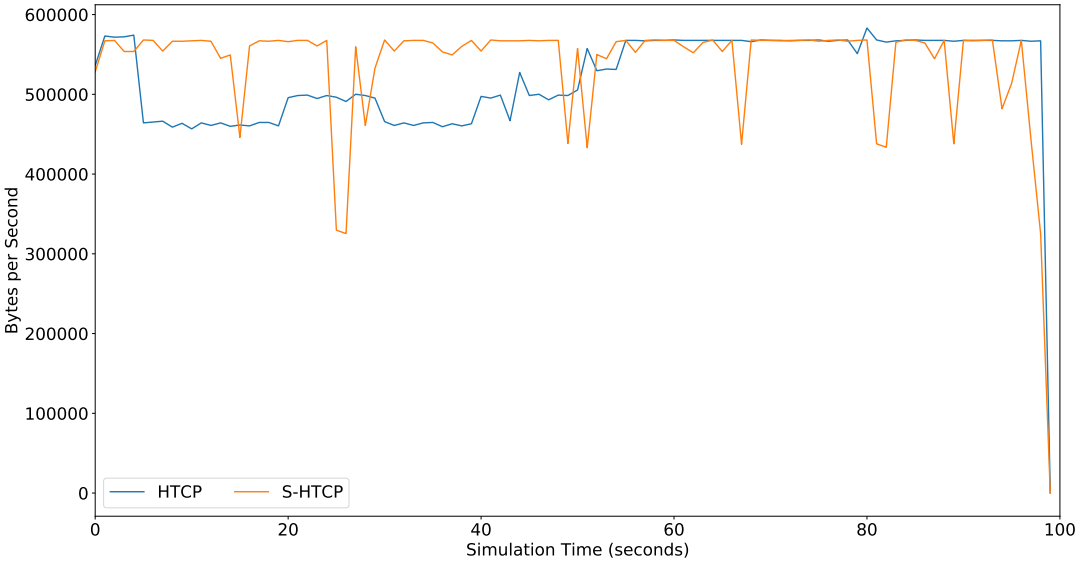


Figure 13: Throughput dynamics for HTCP and S-HTCP

Continuing, Figure 12 offers further insight into how the congestion window behaves in S-HTCP and its predecessor. It becomes clear that the variability of the congestion window for the newly proposed S-HTCP is less than that of HTCP. This can be attributed to the fact that the proposed congestion control algorithm considers the variance in minimum RTT, along with the time elapsed since the last congestion event to conjointly optimize the congestion window, thus leading to fewer congestion events to begin with, as well as a smoother transition between congestion window values due to the near-exponential decay factor introduced to the α and β factors. This conjoint optimization, along with the significantly higher achieved throughput, indicates that this proposed variant's congestion control algorithm can achieve better resource utilization via a more "context-aware" approach, while the average congestion window of S-HTCP is consistently greater than that of HTCP, with reduced variance.

Furthermore, Figure 13 visualizes the throughput achieved by the HTCP and S-HTCP variant, on a per-second basis. The performance of S-HTCP is initially superior to that of HTCP, and does not regress from its maximal value until the single major congestion event around the 25 seconds mark. As a general remark, HTCP initially offers reduced throughput, and proceeds to slowly approach its maximal value at the 55 seconds mark. However, S-HTCP has a better overall, yet not as consistent performance, which sets it almost 4% ahead of its counterpart.

In a similar manner, Figure 14 visualizes the rate at which the two examined TCP variants behave in terms of packet drops, again, on a per-second basis. S-HTCP slightly outperforms HTCP, having a total of 74 packet drops, compared to the 77 such events, observed in the case of HTCP. As a general remark, our proposed modification to HTCP's AIMD mechanism proved consistently less faulty post-stabilization at the 25 seconds mark, which when overlapped with Figure 13 is evidently the point at which the major congestion event occurs for S-HTCP.

Interpolating the data obtained from Figure 13 and Figure 14, it can be deducted that while the number of observed packet drops per second in the case of S-HTCP is measurably and constantly lower, the throughput is more volatile than in the case of classical HTCP, yet on average greater. This behaviour can be partly attributed to the fact that the modified AIMD parameters of S-HTCP render it more likely to create a spike in the reduction of its transmission rate, in the event of packet drop; effectively, the more aggressive approach of S-HTCP also renders it more susceptible to defensive overreactions. However, this effect is mildly mitigated by the protocol's approach in implementing changes (i.e., usage of the exponential decay function). A focal point for future research is the manner in which the alpha and beta parameters' weights interact with each other during severe congestion events, and how that impacts the packet delivery rate and throughput for the entire network. Currently, the results point to the conclusion that this new approach supports the network in reaching the global optimum and maximizing energy efficiency through better transmission scheduling. It is important to benchmark transport-layer protocols' throughput in congestion events, as resilience in such environments is a good indicator of a protocol's capacity to recover from packet drops (20). Similarly, there exist AIMD algorithm implementations that consider not only packet drop rate, but also path quality as a congestion window modifying metric using data from the MAC layer (21).

5. Discussion

This section is dedicated to the throughout analysis of the results obtained from the simulation conducted in the context of this research, as well as the elicitation of useful remarks from our own proposed TCP variant. Information obtained through the analysis conducted in Section 3 is interpolated, to output delicate performance-based recommendations. By considering the variance of the congestion window size, extracted from the visualization of the cumulative moving average of consecutive congestion window's absolute differences, and interpolating our findings with the obtained information on processed load and throughput we deduct that:

- In high-mobility scenarios, the degree of variance of the congestion window is directly proportional to processed load and throughput.
- In high-mobility scenarios, the degree of variance of the congestion window is inversely proportional to average congestion window.
- In high-mobility scenarios, the degree of responsiveness is directly proportional to the achieved PDR.
- In high-mobility scenarios, the degree of responsiveness is inversely proportional to the average congestion window.

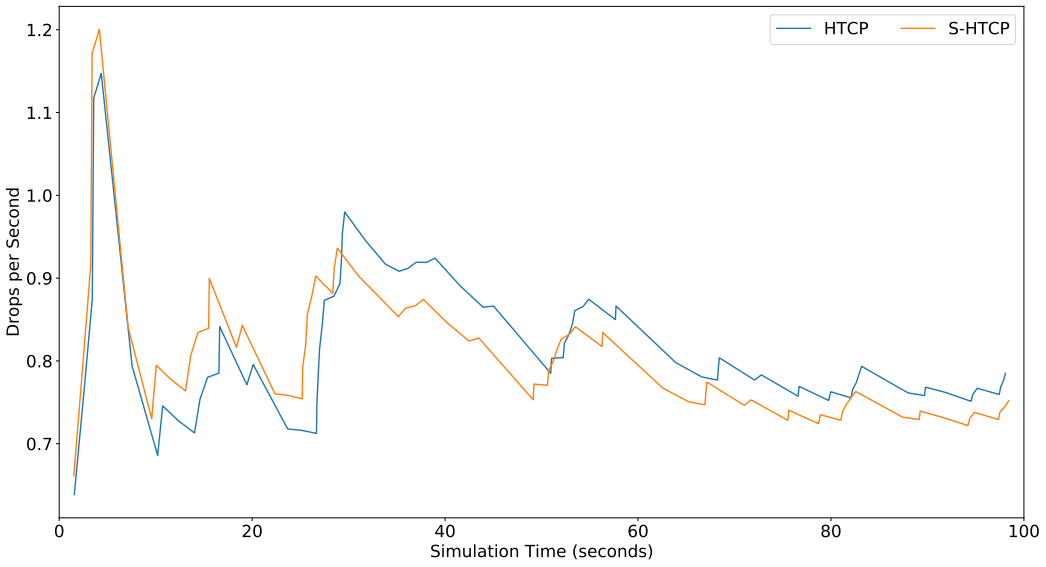


Figure 14: Observed packet drop events per second for the duration of the communication for HTCP and S-HTCP

Regarding the first observation, prime examples of TCP variants where the congestion window variance is proportional to achieved throughput and processed load include HTCP, and High Speed. This can be attributed to the fact that the faster the congestion window changes, the faster it can adapt to the current congestive state and thus achieve greater overall throughput by iterating through the offered load more quickly. The second observation is validated by CUBIC and Vegas, where the average congestion window increases, as its degree of variance increases. This can be attributed to the inherent behaviour of nearly all TCP variants, as they increment their congestion window additively; the more each congestion window alternates between its minimum and maximum values, the more time it spends in the additive increase state. Consequently, the TCP variants with the least degree of variance do not need to spend as much time in the additive increase state. In mobile, ad hoc communications, this means that maximizing the congestion window's additive increase aggressiveness greatly improves performance. This, after all, is the key attribute of both TCP High Speed and HTCP, both of which proved to be peak performers in terms of throughput, despite re-initiating the additive increase algorithm the maximum number of times. Continuing, the third observation is validated by CUBIC, Vegas, Scalable and BIC, where delay proved to be inversely proportional to the achieved PDR. This observation can be attributed to the fact that as end-to-end delay increases, the RTT of each packet and their corresponding ACKs additively increment. Moreover, given the fact that our communication environment is highly volatile, higher delays may introduce a greater number of packet drops, thus negatively affecting PDR. The fourth and last observation is validated by the same set of the TCP variants: Vegas, CUBIC, Scalable, and BIC. It has been observed that as responsiveness in the communication increases, the average congestion window decreased. This can be attributed to the fact that low end-to-end delays enabled each TCP variant to timely assess the congestive state of the link, and thus take the necessary precautions by readjusting the congestion window, and subsequently activating the Slow Start sequence. From the above remarks, we can deduce that in the examined scenario, the degree at which the congestion window varies, is directly proportional to processed load and throughput yet it also is inversely proportional to average congestion window. Moreover, responsiveness is directly proportional to the achieved PDR and inversely proportional to the average congestion window.

As a final remark, our own developments demonstrated that enabling awareness of the time elapsed since the last congestion event (the Δ metric, discussed in detail in Section 4), alongside the variance of the minimum observed RTT measurably increases performance in terms of responsiveness and throughput, but decreases variance in congestion

window. We also demonstrated the potential of new, well-defined AIMD parameters for delay-sensitive and QoS-demanding ad hoc applications.

6. Conclusion and Future Work

In this paper we have discussed matters directly associated with core attributes of TCP. We analyzed in great detail the functionality and performance-relate attributes of twelve well-known variants of TCP. We engage in a thorough comparative analysis and benchmark the considered TCP variants in terms of congestion window volatility, packet drops on a per-second basis, along with throughput, mean end-to-end delay, jitter, mean congestion window and the standard deviation thereof.

We designed an NS3-based simulation leveraging the Gauss-Markov mobility model, which we also document in detail. Furthermore, we attempt to correlate all our findings and draw conclusions using interpolated information outputted by our comparative analysis. The verdict of our evaluation is that HTCP perform better than the other TCP variants in UAV network conditions. Relying on the specifics of this analysis, we introduce a new TCP variant, S-HTCP, which builds on the characteristics of HTCP to further enhance its performance in a FANET environment. Future developments in this domain shall consider our findings in regards to high-volatility links and their transport-layer attributes and impact on communication efficacy.

Lastly, in future research we will consider the further enhancement of our proposed congestion control algorithm, based on that of the examined HTCP variant. The incorporated efficiency-enhancing elements and processes will place further focus on ad hoc communications, and a Linux implementation of S-HTCP will follow. Additionally, in future research we will investigate the performance of the tested protocols in terms of intra-protocol fairness, as the number of parallel coexisting flows increases, especially in the case of different multipath congestion control algorithms, such as loss-based, and/or delay-based ones.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016941.

References

[1] W.-H. He, Z.-H. Ge, and Y.-P. Hu, "Optimizing UDP Packet Sizes in Ad Hoc Networks," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1617–1619, 2007.

[2] V. Jacobson, "Congestion Avoidance and Control," in *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, (New York, NY, USA), p. 314–329, Association for Computing Machinery, 1988.

[3] S. U. Shenoy, M. S. Kumari, U. K. K. Shenoy, and N. Anusha, "Performance analysis of different TCP variants in wireless ad hoc networks," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 891–894, 2017.

[4] K. G. Tsiknas, K. E. Zoiros, and T. D. Lagkas, "Performance analysis of high-speed TCP protocols in LTE X2 handover under realistic operational conditions," *Telecommunication Systems*, vol. 77, pp. 655–669, Aug 2021.

[5] S. Trivedi, S. Jaiswal, R. Kumar, and S. Rao, "Comparative performance evaluation of TCP Hybla and TCP Cubic for satellite communication under low error conditions," in *2010 IEEE 4th International Conference on Internet Multimedia Services Architecture and Application*, pp. 1–5, 2010.

[6] S. Floyd, "RFC3649: HighSpeed TCP for Large Congestion Windows," 2003.

[7] S. Floyd, S. Ratnasamy, and S. Order, "Modifying TCP's Congestion Control for High Speeds," 06 2002.

[8] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," 03 2004.

[9] L. Brakmo and L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[10] R. Morris, "Scalable TCP congestion control," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 3, pp. 1176–1183 vol.3, 2000.

[11] C. P. Fu and S. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, 2003.

[12] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *IEEE INFOCOM 2004*, vol. 4, pp. 2514–2524 vol.4, 2004.

[13] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, p. 64–74, jul 2008.

[14] A. Baiocchi, A. Castellani, and F. Vacirca, "YeAH-TCP: Yet another highspeed TCP," 2007.

[15] A. Dell'Aera, L. Grieco, and S. Mascolo, "Linux 2.4 implementation of Westwood+ TCP with rate-halving: a performance evaluation over the Internet," in *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, vol. 4, pp. 2092–2096 Vol.4, 2004.

[16] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, (New York, NY, USA), p. 287–297, Association for Computing Machinery, 2001.

[17] M. Allman, V. Paxson, and W. Stevens, "RFC2581: TCP Congestion Control," 1999.

[18] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks," valuetools '06, (New York, NY, USA), p. 55–es, Association for Computing Machinery, 2006.

[19] D. J. Leith, R. N. Shorten, and Y. L. Hamilton, "H-tcp : A framework for congestion control in high-speed and long-distance networks," 2005.

[20] Z. Yue, X. Zhang, Y. Ren, J. Li, and Q. Zhong, "The performance evaluation and comparison of TCP-based high-speed transport protocols," in *2012 IEEE International Conference on Computer Science and Automation Engineering*, pp. 509–512, 2012.

[21] J. Zhao, C. Xu, J. Guan, and H. Zhang, "AIMD-PQ: A path quality based TCP-friendly AIMD algorithm for multipath congestion control in heterogeneous wireless networks," in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1678–1683, 2015.