TCP Cubic Implementation in the OMNeT++ INET Framework for SIoT Simulation Scenarios

Ioannis Angelis[1(✉)], Athanasios Tsipis[2], Eleni Christopoulou[1], and Konstantinos Oikonomou[1]

[1] Department of Informatics, Ionian University, 49100 Corfu, Greece
{iangelis,hristope,okon}@ionio.gr
[2] Department of Digital Media and Communication, Ionian University, 28100 Kefalonia, Greece
atsipis@ionio.gr

Abstract. TCP is a well-known protocol for reliable data transfer. Although TCP was originally designed for networks with low Round Trip Time (RTT) and low error rates over the communication channel, in modern networks these characteristics vary drastically, e.g., Long Fat Networks are usually attributed a high Bandwidth Delay Product. When considering satellite communications, which are also characterized by high error rates but are considered a driving force for future networks, such as the Satellite Internet of Things (SIoT), it becomes clear that there exists an ever-growing need to revisit TCP protocol variants and develop new tools to simulate their behavior and optimize their performance. In this paper, a TCP Cubic implementation for the OMNeT++ INET Framework is presented and made publicly available to the research community. Simulation experiments validate its expected behavior in accordance with the theoretical analysis. A performance comparison against the popular TCP NewReno is also performed to evaluate TCP Cubic's applicability to satellite environments. The obtained results testify to the latter's superiority in efficiently allocating the bandwidth among the different information flows with vast gains to the overall system throughput, thus, rendering it the better candidate for future SIoT environments.

Keywords: INET Framework · Network Simulation · OMNeT++ · SIoT · Satellite Communication · TCP Cubic

1 Introduction

The Internet of Things (IoT) has impacted our lives in many different domains, from everyday use to industrial appliances [6]. Nowadays, there is a growing talk about how Smart Cities, Smart Agriculture, and Smart Grids can better our lives by installing IoT devices (e.g., sensors) that can collect massive amounts of data from the environment in order to assist decision-making processes and

H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 561, pp. 18–37, 2024.
https://doi.org/10.1007/978-3-031-54521-4_2

enable reduction of costs, object tracking, real-time monitoring, etc. Nevertheless, the operation of such devices necessitates the preexistence of a suitable network infrastructure that will facilitate their seamless communication, adhering to specific and stringent network requirements [30], e.g., low latency and bandwidth utilization.

In remote locations where the network infrastructure is not readily available or is considered expensive to build, satellites can play a key role in providing global coverage and connecting remote regions to the Internet. Moreover, satellite communication has become substantially cheaper today when compared to the previous decade, due to the booming of the space industry [10], thus paving the pay for the Satellite IoT (SIoT).

SIoT, as a natural expansion to the terrestrial networks, is characterized by the combination of conventional IoT technologies with satellite communication [22]. As such, it can be exploited for both industrial and commercial use, offering new ways for enhancing remote connectivity and availability of services and applications, especially in critical or high-risk settings where there is a need for continuous and uninterrupted monitoring of the underlying conditions [13], such as wildfire detection in forest regions, disaster or crisis management in machine-to-machine communications, military applications in remote tactical geographical areas, collaborative services in industrial systems, etc.

Still, the use of satellites as intermediate network nodes has a twofold impact on the network. First, wireless communication over long distances experiences a high error rate. Second, the long distance between the Earth and the satellite unavoidably introduces a high Round Trip Time (RTT) and at the same time exhibits a large Bandwidth Delay Product (BDP). These characteristics affect the operation of transport protocols and hinder the SIoT devices' interoperability [12], especially when multiple devices must collectively participate in data acquisition and information exchange or collaboratively support the decision-making process and produce actionable outcomes.

One of the most widespread protocols for reliable and orderly communication is the Transmission Control Protocol (TCP) [26]. It is well known that there exist multiple variations (also called flavors) of the TCP depending on the particular environment. However, for any real-world protocol application to be successful, prior to its standardization, it requires extensive and comprehensive experimentation under simulation environments that accurately mimic realistic conditions and acutely capture the intricacies of diverse network settings.

Having this in mind, in the current ongoing work, we consider the TCP Cubic [19] as the TCP flavor more suitable for SIoT environments, in order to test hypotheses and validate use-case scenarios for future SIoT environments. However, to our knowledge, this is the first publicly available implementation for TCP Cubic using the INET Framework for the OMNeT++ simulator. Inspired by the fact that, at the time of writing, no implementation of the TCP Cubic is openly available for the OMNeT++ discrete event simulator [32], which is one of the most widespread simulators available for distributed networking systems, including IoT, we hereinafter present initial results, from both a theoretical and

practical standpoint, regarding TCP Cubic's implementation in the aforementioned simulator. Our vision is for the presented implementation to become part of the researchers' technology arsenal, an important collaboration tool for experimentation purposes, and ultimately enrich ongoing and future research endeavors, that study the behavior of the SIoT ecosystem, with valuable insights.

1.1 Contribution

Based on the preceding, our contribution to the research community is fourfold and can be summarised in the following aspects:

- Motivated by the current absence of the TCP Cubic flavor in the OMNeT++ simulator, in this paper, its implementation in the INET Framework is presented and made publicly available to the research community.
- An overview of the TCP structure within the INET Framework is also provided to support the logic of our implementation, with detailed descriptions relating to its class, subclasses, and functions, along with an analytical study of its idiosyncrasies.
- Additionally, comprehensive simulation experiments are conducted to evaluate the expected behavior of our model under two different topologies; that is, in point-to-point and dumbbell networks. The results are in accordance with the theoretical analysis.
- Finally, to test TCP Cubic's applicability to satellite topologies (i.e., in SIoT systems), a comparison with the well-known TCP NewReno, which is used as a baseline here, is conducted for a variety of RTT values. The results clearly indicate how the former surpasses the latter, with notable gains in throughput, making it the more suitable candidate for augmenting communication in future SIoT environments.

1.2 Paper Structure

The rest of the paper is organized as follows: Sect. 2 includes necessary background information on the TCP, OMNeT++ simulator, and INET Framework; Sect. 3 describes the theoretical behavior of TCP Cubic; Sect. 4 presents the structure of TCP in INET as well as how it is extended with the implementation of TCP Cubic; Sect. 5 demonstrates simulation results of TCP Cubic's performance and provides insights from its comparison against the TCP NewReno in SIoT environments; and, finally, Sect. 6 concludes the paper, summarizing its key findings and providing future research directions.

2 Background Information

Following, some background information on TCP and OMNeT++/INET Framework is provided to familiarize the reader with the topics of the current research investigation.

2.1 TCP Related Work

TCP Cubic is the default implementation on the Linux kernel, its appearance tracing back to 2008 [17]. As the researchers highlight in their work, a path with a 10 Gbps bandwidth, an RTT of 100 ms delay, and a segment size of 1250 bytes produces a high BDP that standard TCP variations like Reno and NewReno will take approximately 1.4 h when growing their window to the full BDP size, a fact that severely under-utilizes the network path link. When referring to satellite communications, high BDP is to be expected depending on the altitude of the orbit. For instance, low-Earth orbit satellites located 160 km to 1000 km above the surface of the Earth result in RTT smaller than 100 ms. On the other hand, Geosynchronous Equatorial Orbit satellites are positioned at 35.786 km over the ground and hence their RTT can range anywhere between 480 ms to 700 ms.

TCP Hybla, on the other hand, was introduced in 2004 for satellite communications as a variant to the TCP NewReno [19]. Hybla possesses a higher throughput than TCP NewReno given the same values for their BDP and RTT, however, it is unfair to other flows that use the same algorithm [8].

TCP-START [23] is another variation of TCP that has been designed for the Satellite Internet around 2006. The authors proposed three new mechanisms to overcome the drawbacks of the performance of standard TCP in Satellite Environments: 1) Congestion Window Setting: This mechanism helps to avoid the unnecessary reduction of the transmission rate using Available Bandwidth Estimation (ABE) when a bit error causes the data loss. 2) Lift Window Control: As its name suggests, this mechanism increases the cwnd faster using the values of TCP Reno and the ABE. 3) Acknowledgment Error Notification: This mechanism is used to minimize unnecessary timeouts by ACK loss or delay, and avoid the reduction of throughput via mis-retransmission of data. The experimentation of the authors showed to have better throughput from TCP-WestwoodBR and TCP-J in addition to having similar fairness with TCP-WestwoodBR in homogeneous environments and slightly worse results in heterogeneous environments. This TCP variation requires only changes on the sender side.

TCP Peach [4] was designed for satellite networks and uses four algorithms, two of which are Fast Retransmit and Congestion Avoidance. It also introduced two new algorithms, namely the Sudden Start and Rapid Recovery. TCP Peach tries to distinguish the drops emanated by congestion from those originating due to link failures. With the new algorithms, TCP Peach can probe the available bandwidth in one RTT using dummy packets with low-priority service that do not contain new data for the receiver. An improvement to the particular protocol is the so-called TCP Peach+ [5] which replaces the low priory dummy packets with low-priority new data.

TCP Norirdwick [28] was introduced in 2008 as an alternative congestion control algorithm that uses burst base transmission. The researchers in their paradigm use this algorithm to transfer sort web traffic over satellite links. The newest version of this protocol is the TCP Wave [2] which can be used in more network-wide communication scenarios. It is designed with three principles [1]: Burst Transmission, ACK-based Capacity, and Congestion Estimation, and Rate

Control algorithm. This protocol requires dramatic changes to the side of the sender.

In the last years, a new congestion control has been developed, TCP Bottleneck Bandwidth and Round-trip time (BBR) by Google, as an alternative algorithm for TCP Cubic. TCP BBR is used extensively in the servers of Google after 2017 [9]. The algorithm estimates periodically the minimum RTT and the available bandwidth of the path, its main goal being to not build up queues in the intermediate nodes. In a follow-up work [33] it was shown that TCP BBR dominates the TCP Cubic on start-up and steady state in satellite environments. Conversely, in another work [11], TCP Hybla was shown to perform better for small-size downloads (e.g. Web Pages) relative to TCP BBR.

Considering TCP as the prime transport protocol for SIoT environments, the next step is experimenting through simulation. As presented, a lot of interesting TCP flavors have been proposed in the literature for satellite communication that lack, however, implementation within the OMNeT++/INET Framework. This is also the case for TCP Cubic. Given that i) a growing volume of related research uses TCP Cubic as their preferred flavor for SIoT systems; ii) the fact that it is the default variant in the Linux distribution; and iii) there is no such implementation currently publicly available in OMNeT++, our aim in the upcoming sections is to develop one within the INET Framework and make it openly accessible to the research community.

Note that TCP Peach and Peach+, as discussed, are also popular solutions for SIoT systems. However, their functionality necessitates routers to have first implemented some queue priority scheduling disciplines that, in turn, require extensive modifications on the receiver side. In contrast, the implementation of TCP Cubic, as provided here, is embedded seamlessly within the existing TCP family of INET by overriding some of its current TCP classes, thus, making it a more suitable addition to the OMNeT++ simulator.

2.2 OMNeT++ and INET Framework

Before proceeding with our proposed implementation, it is necessary to introduce the reader to the selected simulator. Objective Modular Network Testbed in C++ (or OMNeT++[1] in short) is an open-source modular component-based simulator for networks, that increasingly gains momentum in the academic community for diverse networking research agendas [31]. It essentially comprises a discrete, event-driven, and general-purpose simulator that gives the flexibility to simulate different types of networks like peer-to-peer, queuing, ad-hoc, and many more [32].

The INET Framework[2] on the other hand, is a popular open-source library for the OMNeT++ simulator [21]. It provides a variety of protocols and modes to simulate wired and wireless networks. Further, it includes a model for the simulation of the complete Internet stack.

[1] Accessible at the web address: https://omnetpp.org/.
[2] Accesible at the web address: https://inet.omnetpp.org/.

One of the protocols, offered by the INET Framework, is that of TCP. Using the modular approach of OMNeT++, INET implements the core functionality of TCP wherein different congestion control algorithms extend the "TcpBaseAlg" according to their needs. With that said, currently, there are implementations for the following algorithms: Tahoe, Reno, NewReno, Vegas, Westwood, and DCTCP. In this paper, aiming to enrich the toolkit of available TCP algorithms, the authors provide the implementation of TCP Cubic as an extension to the TcpBaseAlg, and then present initial results from its application in the SIoT ecosystem.

3 Theory of TCP Cubic

The main features of TCP Cubic are scalability, stability, and its proclivity towards fairness in regard to competing flows that use the same path [7]. To do so, TCP Cubic redefines the congestion avoidance algorithm and the computation of the slow start threshold (*ssthresh*) but uses the same mechanisms for Fast Retransmit and Fast Recovery as the TCP NewReno after a package loss. Moreover, in the TCP Cubic new state variables for the computation of the congestion window (*cwnd*) are introduced. In particular, when a loss event occurs the TCP Cubic registers the maximum value of *cwnd* in the variable W_{max} before the reduction of *cwnd*. After performing multiple decreases of *cwnd* by a constant factor β, it enters the congestion avoidance phase when the Fast Recovery phase has ended.

Going into more detail, TCP Cubic uses *epoch_start*, t, and K to track the time from the loss event. The variable *epoch_start* is used to register the time when TCP for the first time acknowledges (ACK) new data. The t variable keeps track of the time that has passed from the loss event. K stands for the time when the *cwnd* reaches the value W_{max}, if no further losses occur, and is calculated as

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}, \tag{1}$$

where C is a constant that regulates the aggressiveness of TCP Cubic.

With the previous variables, Cubic is able to determine the current *cwnd*, according to the following expression:

$$W(t) = C(t-K)^3 + W_{max}. \tag{2}$$

The cubic function in the right part of Eq. (2) is utilized by TCP Cubic for the computation of the window growth function of *cwnd*, relative to W_{max}, in the congestion avoidance phase.

Different from most alternative congestion control algorithms to Standard TCP, which increases the *cwnd* using convex functions, TCP Cubic uses both a concave and a convex profile for the increase of *cwnd* in the congestion avoidance [17]. The alteration between the two profiles helps the TCP Cubic to achieve higher network utilization and stability. This is done as follows. Before TCP

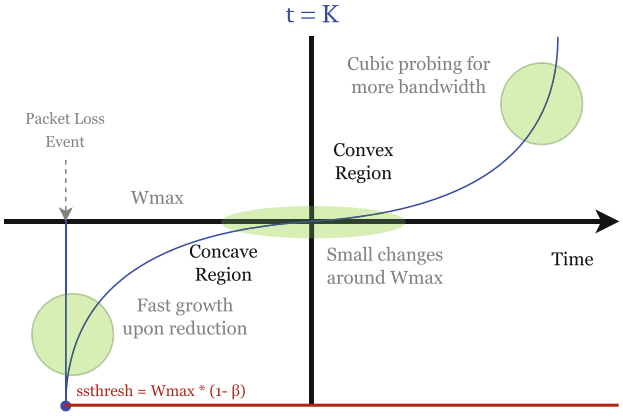


Fig. 1. Cubic theoretical growth function depicting its concave-convex profile switching behavior.

Cubic performs multiple decreases due to packet loss, it registers the current *cwnd* to W_{max}. Then it enters the congestion avoidance where the *cwnd* growth follows the concave profile until it reaches the W_{max} used as a plateau. In this region (i.e., the concave region), *cwnd* initially grows fast and as it approaches the W_{max} plateau its changes become very small. When it reaches the W_{max}, the profile switch takes place whereby the *cwnd* enters the convex region. In this region, the increase in the growth of *cwnd* is initially slow, but as time progresses it speeds up in order to probe for more bandwidth.

The aforementioned theoretical behavior of the window adjustment is more accurately depicted in Fig. 1. In the beginning, after the packet loss event occurs, *cwnd* experiences a fast growth as it has only passed a small amount of time since *epoch_start*. As time passes and t approaches K, the increase to *cwnd* becomes smaller. At some point when the t will be larger than K, the Cubic turns from the concave to the convex profile. In the convex region, the Cubic updates the *cwnd* growth initially by small values. However, as time t passes, the changes grow continuously larger maximizing in this way bandwidth probing.

When Standard TCP detects a loss event from Retransmission Timeout (RTO) or three duplicate Acknowledgments, it computes the new *ssthresh* as half of the current *cwnd* size. This approach is too conservative and dramatically reduces the performance of TCP. In contrast, TCP Cubic reduces the current *cwnd* by a factor β that is smaller than one-half of the one that the Standard TCP uses. Typically, this factor is set to 0.2 or 0.3 [17,27]. Cubic then uses the below expression (3) to calculate the new value of *ssthresh* as:

$$ssthresh = W_{max}(1 - \beta). \tag{3}$$

The TCP Reno family, on the other hand, works very well in networks with small RTT and small BDP. TCP Cubic uses the TCP-friendly region to achieve the same throughput as the TCP Reno family for those networks. This is done by

computing the theoretical *cwnd* ($W_{tcp(t)}$) of the TCP Reno family at the reception of each new Acknowledgment in the Congestion Avoidance phase. Next, it compares the $W_{tcp(t)}$ with the current *cwnd*. If *cwnd* is found to be less than $W_{tcp(t)}$, the protocol sets *cwnd* equal to $W_{tcp(t)}$ at each ACK reception, where the $W_{tcp(t)}$ is computed by Eq. (4):

$$W_{tcp(t)} = W_{max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}. \tag{4}$$

TCP Cubic utilizes a heuristic sub-routine to improve the convergence speed. While incoming flows join the same link path, existing flows must release some of their bandwidth shares to accommodate the newly arrived. This is done when TCP detects a packet loss event. In such cases, if the *cwnd* is smaller than the previous W_{max}, then it assigns to the W_{max} a smaller value than the value of the current *cwnd*. Otherwise, the W_{max} registers the actual value of the *cwnd*. The end goal of the fast convergence sub-routine is to give some time to new flows to catch up with the existing *cwnd* of the other flows within the network.

4 INET Implementation

INET Framework is an open souse library that provides simulation models of the Internet Stack for wire and wireless networks. These models follow the modular approach of OMNeT++, wherein smaller basic components build compound and more complex models. For example, INET provides basic queues that can be used to form a buffer for the application. In this section, we will describe the details of our implementation of TCP Cubic using the INET Framework.

4.1 TCP Structure on INET

TCP is a complex protocol to simulate due to its mechanisms used to provide for reliable, in-ordered, and error-checked delivery of information. In Fig. 2, the structure of the TCP model in the INET Framework is presented with a UML class diagram.

The TCP class is composed of several (sub-)classes such as TcpSendQueue, TcpReceiveQueue, and TcpConnection. To administrate the incoming and outcoming chunks of information, TCP calls TcpSendQueue and TcpReceiveQueue to manage the corresponding buffers accordingly. TcpConnection follows the functionality guidelines set forth by RFC 793 [26] to implement the state machine that handles all the connections that open and close from the host. If Selective Acknowledgments (SACK) are used, TcpConnection calls the TcpSackRexmitQueue class for the handling of SACK retransmission.

The TcpAlgorithm class is an abstract class that encapsulates all behavior of the transfer state of TCP (e.g., functions for establishing and then closing connections). TcpBaseAlg inherits and extends TcpAlgorithm class, and it is responsible for keeping track of the TCP timers, in addition to restarting them when the need arises, as well as for the collection of appropriate statistics. Finally,

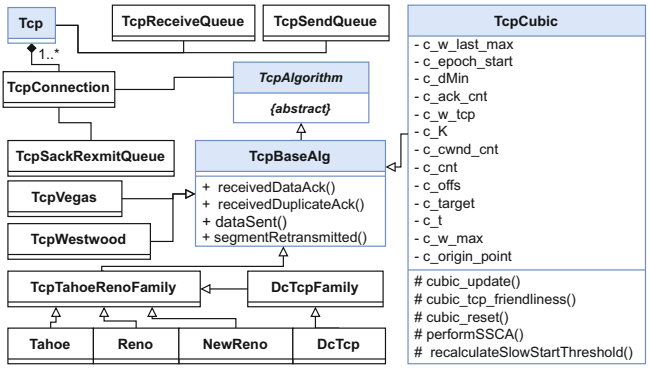


Fig. 2. UML diagram of the TCP on the INET Framework along with the TcpCubic class implemented in this paper.

it provides a general implementation for tasks relevant to data sending, to ACKs or duplicate ACKs reception, and to what action must be taken given out-of-order segment retransmission.

Over time, a variety of congestion avoidance mechanisms have been created for TCP that use different strategies to deal with congestion. Some of them adopt the same logic and elements. For this reason, a family class has been implemented in INET that provides the basic utility for those algorithms. The family class is inherited from the TcpBaseAlg class and is used to facilitate the easier management of the corresponding mechanism algorithms. To exemplify this, consider the Tahoe, Reno, and NewReno TCP flavors all of which inherit characteristics from the same family, i.e., the TcpTahoeRenoFamily.

In some cases, the TCP flavors can directly inherit and overwrite the functions of the TcpBaseAlg class, as has been done with TCP Vegas and Westwood. Even though TCP Cubic uses some of the characteristics of TCP NewReno, it was decided to directly inherit and overwrite the TcpBaseAlg class for our INET TCP Cubic implementation, as shown in Fig. 2.

4.2 TCP Cubic Implementation

In this section, we will deepen the analysis of our TCP Cubic class implementation, henceforth named TcpCubic, as well as the functions we use to define its behavior. We have used the implementation of Cubic in Linux code as a guideline. The code of Linux uses integer arithmetics and kernel functions for performance tuning. Conversely, the INET Framework is a simulation library within OMNeT++ and not a kernel per se. For this reason, we will not focus on TCP Cubic state variables of the Linux code, although comprehensive documentation on these aspects can be found by other researchers [15,20]. For dissemination purposes, our TCP Cubic implementation is made publicly available

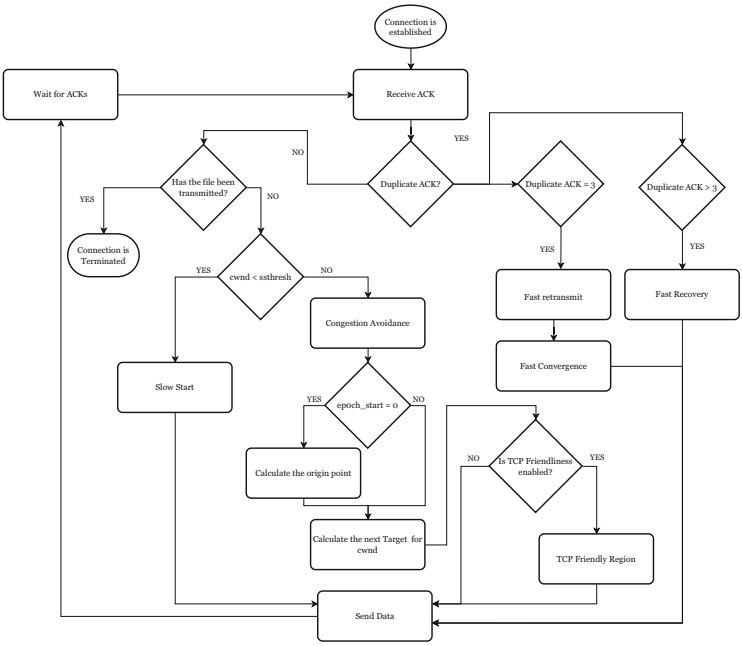


Fig. 3. Flow chart of TCP Cubic.

on the Github repository[3] Its flow chart (with the exception of timeouts which are part of the core TCP) is illustrated in Fig. 3 while its functions are discussed thoroughly below.

TPC Cubic External Functions: Based on the previously defined TCP Cubic functionality, the TcpBaseAlg class is overridden by the following functions:

dataSent(): This function is called each time the TCP sends a new segment. It schedules the retransmission timer if it is not running, starts the measurement of RTT, and remembers the last time that data was sent. INET doesn't support TCP timestamps but instead uses a utility class, called TcpSegmentTransmitInfoList, which stores information about the packet like the first and last time the package was sent and the number of transitions. With this information, we can calculate later the time for the variables t and $dMin$.

segmentRetransmitted(): This function is called from the TcpConnection class whenever needed, after data retransmission. Similar to the dataSent() function, we need to update the time in TcpSegmentTransmitInfoList when we retransmitting a package.

receivedDataAck(): This class is called each time the TCP receives an ACK from the data that was sent. The TcpBaseAlg class provides basic functionality

[3] Accessible at the web address: https://github.com/GIANNIS-AGGELIS/INET-TCP-CUBIC.

for retransmission handling as well as a PERSIST timer, while leaving to each flavor of TCP the decision of determining the specific action that will happen upon reception of each ACK. For TCP Cubic we extend this function to calculate the $dMin$ using the TcpSegmentTransmitInfoList class. At the same time, the action of Fast Recovery is implemented here whereas for performing Slow Start or Congestion Avoidance a call is made to the function performSSCA(). In the end, the sendData() function is called and TCP will try to send a new date if it is allowed from the receiver window.

receivedDuplicateAck(): As the name suggests, this function is called when the TCP receives a duplicate ACK. It also implements the Fast Retransmit algorithm, which calls the recalculateSlowStartThreshold() function for the calculation of $ssthresh$ whenever in need. Finally, it performs the action of Fast Recovery when it receives more than three duplicate ACKs.

TPC Cubic Internal Functions: Following, a detailed description of the internal functions of TcpCubic is provided:

cubic_reset(): This function is called to reset the variables of TCP Cubic during a time-out.

recalculateSlowStartThreshold(): It constitutes a utility function that is called when TCP needs to calculate the new $ssthresh$ value according to Eq. (3). Also, this function includes the implementation of the fast convergence heuristic subroutine.

cubic_update(): This function is called by performSSCA() each time TCP receives a new ACK in the Congestion Avoidance phase. For its implementation, we have closely followed as a guideline the pseudo-code provided in the original work of Ha et al., [17]. In INET, the default *cwnd* size is represented in number of bytes. Thus, for the calculation of both K and the target window, one must first convert the *cwnd* size into a number of segments in order to properly proceed with its calculation. One observation that we make is that we need to compute the absolute value of $|t - k|$ from Eq. (2).

cubic_tcp_friendliness(): This function is called at the end of cubic_update() with the aim of computing the theoretical *cwnd* of the TCP Reno family. In the circumstance that *cwnd* is smaller than the W_{tcp}, then it uses that as the target window.

5 Simulation Results

In this section, we discuss the different simulation experiments that were conducted to validate our INET implementation of TCP Cubic. In all experiments, we simulate an FTP scenario. Two sets of experiments are considered. For the first, we evaluate our implementation under generalized client-server scenarios whereas, for the second, we consider a more specific scenario targeted at the SIoT environments.

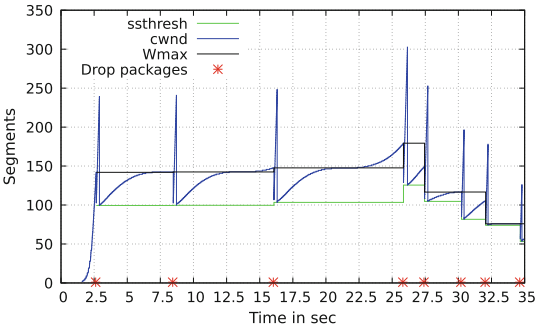


Fig. 4. Growth of *cwnd* of TCP Cubic over time.

5.1 Evaluation Under Client-Server Scenarios

For the first experiment, we begin with a typical use case by employing two Standard Hosts from the INET library which are connected directly via a wire link in a point-to-point communication. One node represents the Client and the other the Server with the first running a Tcp Session App and the second running a Tcp Sink App respectively. The file size that we wish to transfer is 20 MB, and the maximum segment size is 1024 bytes. The transmission begins at 1.2 s. For link characteristics, we choose a one-way delay at 40 ms, a 5 Mbps of data rate, and as for the error rate, we input a probability of 0.0005. The last is intentionally set to a low value for the initial experiment since this yields a small number of errors on the link, allowing for clearer observations of the behavior of *cwnd* after a loss event.

In Fig. 4, in addition to the growth of *cwnd* over time, we have plotted the values of *ssthresh* and the W_{max}. The link in this case has produced eight (8) errors in packages that contain data. The spikes on *cwnd* that are created after a packet loss are the result of the Fast Recovery algorithm. After the three (3) duplicated ACKs, Fast Recovery inflates the *cwnd* by one package for each additional duplicate ACK. When the TCP acknowledges new data, it deflates the *cwnd* to the value of *ssthesh*, and the algorithm for Congestion Avoidance algorithm kicks in.

In the first congestion epoch, the Congestion Avoidance increases the *cwnd* from *ssthresh* only with the concave profile to W_{max} as a new loss was detected in the same *cwnd* as the loss. The *ssthresh* and W_{max} for the second congestion epoch remain the same but this time the *cwnd* surpasses the current W_{max} and starts slowly to increase until the next packet loss. In the third congestion epoch, we can see the full evolution of *cwnd* from convex to concave region up to the next packet loss. The following package loss is very close in time and the *cwnd* does not reach the current W_{max}. This behavior repeats, as observed by Table 1, which encapsulates the time t when packages are lost as well as the value of time when the growth of *cwnd* is expected to change from concave to convex if no further losses appear.

Table 1. The first column shows the time that a window reduction has occurred, the second shows the time when *cwnd* is expected to change profile if no further loss occurs, and the third indicates whether a profile switch actually happened.

Packet Loss Recorded Time	Switch Expected Time	Profile Switch
2.90	7.61	NO
8.72	13.42	YES
16.32	21.09	YES
26.16	31.25	NO
27.70	30.70	NO
30.43	34.83	NO
32.24	33.53	NO
34.80	38.59	NO

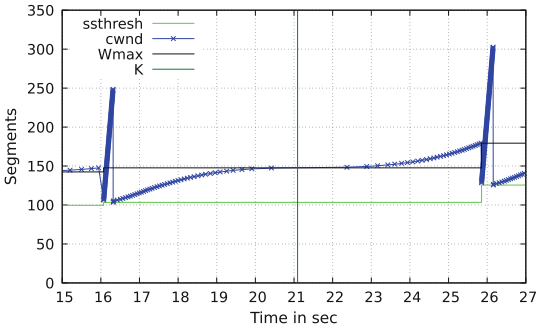


Fig. 5. Switch from concave to convex profile, depicted by the green vertical line. (Color figure online)

To further validate the expected profile switch attitude of TCP Cubic, in Fig. 5 the value of K is depicted by zooming into the third congestion epoch of the previous diagram (Fig. 4). Here, we can clearly witness that the alternating concave and convex behavior of *cwnd* closely follows the theoretical behavior of TCP Cubic as shown in Fig. 5, while *cwnd* stays almost constant for some time around the $Wmax$.

For the second experimental scenario, as a proof-of-concept, we construct a representative dumbbell topology where two routers are located between four Clients and four Servers at the bottleneck between two endpoints. Figure 6 visualizes the exact topology, whereby our goal is to test the TCP Cubic in a shared link. The links that connect the routers with the Servers and the Clients have a 10 Gbps capacity and no error rate. The default configurations of the routers use drop tail queues with a 10.000 capacity of packets. The bottleneck link between the two routers has a bandwidth of 100 Mbs and a small probability of 0.0005 for error rate.

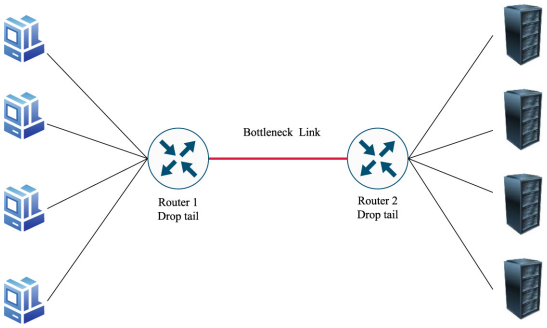


Fig. 6. The considered topology for the typical client-server simulation scenarios.

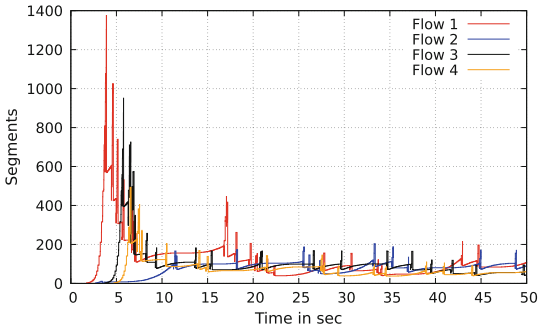


Fig. 7. Evolution of *cwnd* over time from 4 flows under the same bottleneck link.

According to Fig. 6, each of the four Clients is connected to its corresponding Server. We investigate the FTP scenario where each Client transfers a big file to the Server and the simulation runs for 50 s. Each connection opens sequentially with a difference of 1 s from the previous one, starting at 1.2 s. The results from this experiment are illustrated in Fig. 7, where it can be observed that for $t > 10$ s the network status has stabilized and all flows have values of *cwnd* close to one another for the remaining duration of the experiment.

5.2 Evaluation Under SIoT Scenarios

In this section, the aim is to evaluate the TCP Cubic in SIoT environments and contrast it to the one obtained by TCP NewReno. We begin our investigation by considering a simple SIoT topology, as depicted in Fig. 8, where the Clients and Servers are connected to Gateways. The Gateways, in turn, connect the two components of the network with the remote intervention of the Satellite.

Because opening a TCP connection from an IoT device is expensive in resources, and typical IoT devices, such as low-cost sensors, are hardware constrained (e.g. due to limited battery) [29], we assume that the IoT devices are connected to sink nodes, which collect the data and then send it deeper into

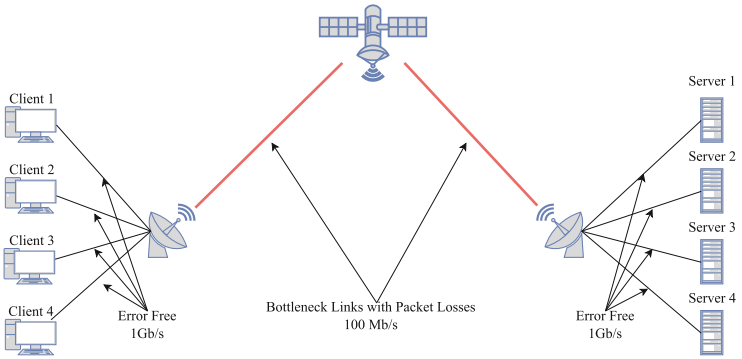


Fig. 8. The considered satellite topology for the SIoT simulation scenarios.

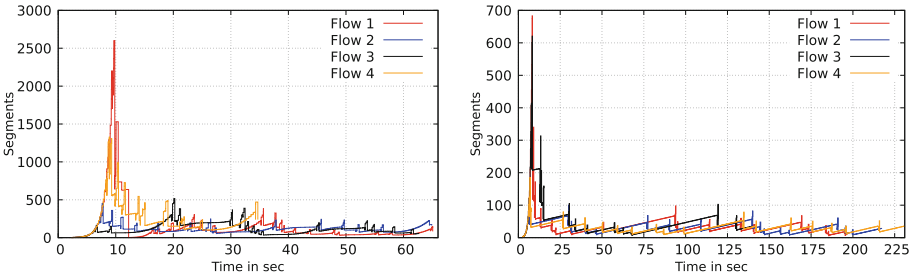
the network, usually towards a centralized computing infrastructure, for further analysis. Sink nodes have more available resources and can open and maintain TCP connections. That being said, for the case presented in Fig. 8, the sink nodes are represented as Clients that have already collected the data from the underlying IoT and, thus, wish to transfer this information to the corresponding Servers via the intermediate Satellite.

In the third simulation scenario, the goal is to test the performance of TCP Cubic over the satellite links. In this case, the network bottleneck manifests at the satellite links shared along the path that connects the Clients with the Satellite and the path that connects the Satellite with the Servers, respectively. The links of the Clients and the Servers with their corresponding Gateways are attributed a data rate of 1 Gbps with no error loss while the links connecting the latter with the Satellite have a 100 Mbps data rate with 0.0005 probability for each link to produce an error.

Following, we compare our implementation of TCP Cubic against the TCP NewReno from the INET library (i.e., Fig. 9). TCP NewReno is considered a baseline for satellite communication [3,25], and although today there exist other TCP flavors that are clearly better equipped to handle satellite networks (e.g., consult Sect. 2 for more information), due to TCP NewReno being the only other one that has already been incorporated in the INET library, the authors decided that the particular protocol is the best available alternative to test the performance of TCP Cubic in OMNeT++.

Under this light, we assume an FTP scenario wherein each Client needs to transfer a file size of 15 MiB, and all Clients start the transfer simultaneously (i.e., at 1.2 s). The one-way delay from the Gateway to the Satellite is 100 ms and so the total RTT between each Client and its Server is 400 ms.

To comprehensively visualize and collate the simulation results, the *cwnd* form is separately plotted as a function of time for the TCP Cubic in Fig. 9a and for the TCP NewReno in Fig. 9b. Evidently, TCP NewReno experiences a substantially worse performance relative to the TCP Cubic as all four flows finish significantly later than their corresponding counterparts for the TCP Cubic case.



(a) TCP Cubic. (b) TCP NewReno.

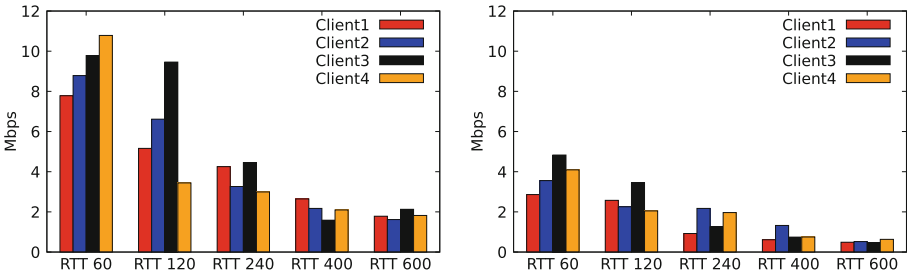
Fig. 9. Comparison of TCP Cubic against TCP NewReno for four flows over the same satellite link.

In fact, two out of its four flows finish after the 200 s time mark, whereas for the TCP Cubic no flow exists that finishes beyond the 70 s. This happens because TCP NewReno reduces the *cwnd* by half each time a package is lost and in the Congestion Avoidance phase it increases the window by $1/cwnd$ for each new ACK, a fact that produces linear growth. The rapid spikes that are observed in Fig. 9b during the beginning are the result of the Fast Recovery algorithm.

Overall, TCP Cubic performs markedly better as all the flows finish much sooner relative to the ones of TCP NewReno. Furthermore, it can be seen that, after 15 s has passed, the network goes into an almost equilibrium state where the flows operate above 150 packages for the whole remainder of the time. It must be noted that, for $t \approx 10$, Flow 1 shows a massive increase in *cwnd*. This is caused by a package loss that appeared much later relative to the other flows and thus the Fast Recovery algorithm receives a large number of duplicate ACKs. Subsequently, around the 12 s time mark, Flow 1 experiences an RTO and, as a result, it reduces the *cwnd* to one package and starts again with the Slow Start algorithm until the new *ssthresh* value is reached, given that no further loss occurs. These observations testify to the ability of TCP Cubic to successfully handle the workload and packet losses in SIoT systems, and better manage the bandwidth allocation among the different flows when compared to TCP NewReno.

To further validate this claim, in the last experimental scenario, given the SIoT topology that was previously presented in Fig. 8, different values on the delay of the satellite links are taken into account, considering a 15 MiB file transfer from each Client to the corresponding Server. Specifically, in this configuration, our aim is to explore the throughput of each Client by enforcing the following RTT values: 60 ms, 120 ms, 240 ms, 400 ms, and 600 ms. Again, the packet loss probability is set to 0.0005 for each satellite link. Figure 10 captures the results of the Clients' averaged throughput, after repeating the experiment for ten independent runs with different seeds for the error rate.

As clearly demonstrated, TCP Cubic (Fig. 10a) has a much higher average throughput than TCP NewReno (Fig. 10b) in all cases. Moreover, for both pro-



(a) TCP Cubic. (b) TCP NewReno.

Fig. 10. Simulation results of the average throughput per Client for the TCP Cubic versus the TCP NewReno.

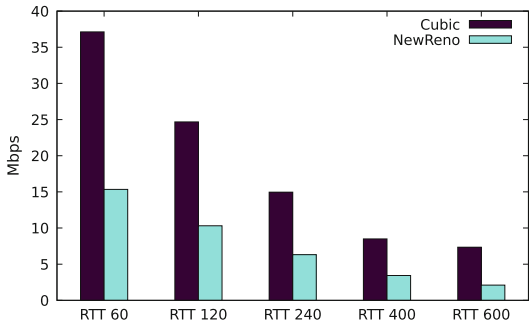


Fig. 11. Average aggregated throughput for TCP Cubic versus TCP NewReno.

tocols, it is observed that the average throughput decreases as the RTT rises. This is to be expected since an increase in the delay also leads to an increase in total transfer time. Of special interest are the cases where the RTT is ≥ 120 ms, as these values are more inherent to satellite environments [18]. Still, even under these realistic circumstances, TCP Cubic is observed to have almost double the throughput for most clients, significantly outperforming the TCP NewReno.

To verify this finding, the final plot (i.e., Fig. 11) illustrates the average aggregated throughput of each protocol for all experiments that were executed per RTT value. Clearly, the employment of TCP Cubic results in substantial throughput gains, outperforming the TCP NewReno by a large margin, TCP Cubic performs at least two times the throughput relative to TCP NewReno. Actually, for the worst case where the RTT equates to 600 ms, the TCP Cubic archives almost 7.5 Mbps while the TCP NewReno approximately 2 Mbps. Furthermore, it is observed that as the RTT grows the difference in the average aggregated throughput between the two protocols also grows, a fact that renders TCP Cubic the better overall candidate for dealing with high latency over long network distances such as those found in the SIoT ecosystem.

As a side note, one must keep in mind that TCP NewReno was not specially designed for Long Fat Networks. On the other hand, TCP Cubic is designed specifically for these types of networks and hence it can be seamlessly integrated in satellite communication environments. To this end, we believe that the implementation presented here can become a valuable addition, for practitioners and researchers alike, to the existing arsenal of network simulation tools. Nevertheless, more tests must be carried out before fully assessing its eligibility for SIoT networks, especially in regard to other TCP variants, besides the TCP NewReno, and in combination with more realistic and complex SIoT topologies. We leave the detailed analysis of these matters to future work.

6 Conclusions

In this paper, using the INET Framework, the TCP Cubic was developed and presented. To the authors' knowledge, this is the first publicly available implementation of TCP Cubic for the OMNeT++ simulator. Under this scope, a detailed description of the TCP protocol class and functions in the INET was provided along with information about how they are extended to fit the current research needs of TCP Cubic. Simulation results, evaluating its behavior, were in close alignment with the theoretical analysis. A direct comparison of the implemented TCP Cubic against the TCP NewReno was also performed, focusing on SIoT environments and concluding that TCP Cubic is the more suitable candidate with notable augmentations to the overall system performance.

For future work, the authors plan to provide an updated version of TCP Cubic that utilizes for the Slow Start phase the HyStart algorithm [16]. HyStart was not part of the original work on TCP Cubic [17] but since then it has been used in conjunction with TCP Cubic under various networking contexts. Hence, it could prove extremely beneficial to investigate their combination in SIoT systems, as many researchers have already seen promising results in this regard [24]. In parallel, it is believed that the use of the ESTNeT simulator [14], which also embeds the INET Framework, can create more realistic simulations of satellite communications. Thus, its synergy with the presented TCP Cubic implementation is viewed as the next logical step towards shedding light on the more intriguing aspects of the SIoT ecosystem. Furthermore, additional experimentation will be conducted, considering real-world scenarios with a large number of IoT devices, where the scalability of the protocol will be thoroughly tested.

References

1. Abdelsalam, A., Luglio, M., Patriciello, N., Roseti, C., Zampognaro, F.: TCP wave over Linux: a disruptive alternative to the traditional TCP window approach. Comput. Netw. **184**, 107633 (2021). https://doi.org/10.1016/j.comnet.2020.107633. https://www.sciencedirect.com/science/article/pii/S1389128620312585

2. Abdelsalam, A., Luglio, M., Roseti, C., Zampognaro, F.: TCP wave resilience to link changes. In: Proceedings of the 13th International Joint Conference on E-Business and Telecommunications, ICETE 2016, pp. 72–79. SCITEPRESS - Science and Technology Publications, LDA, Setubal, PRT (2016). https://doi.org/10.5220/0005966700720079
3. Abdelsalam, A., Roseti, C., Zampognaro, F.: TCP performance for satellite M2M applications over random access links. In: 2018 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–5 (2018). https://doi.org/10.1109/ISNCC.2018.8531048
4. Akyildiz, I., Morabito, G., Palazzo, S.: TCP-Peach: a new congestion control scheme for satellite IP networks. IEEE/ACM Trans. Network. **9**(3), 307–321 (2001). https://doi.org/10.1109/90.929853
5. Akyildiz, I., Zhang, X., Fang, J.: TCP-Peach+: enhancement of TCP-peach for satellite IP networks. IEEE Commun. Lett. **6**(7), 303–305 (2002). https://doi.org/10.1109/LCOMM.2002.801317
6. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010). https://doi.org/10.1016/j.comnet.2010.05.010. https://www.sciencedirect.com/science/article/pii/S1389128610001568
7. Cai, H., Eun, D.Y., Ha, S., Rhee, I., Xu, L.: Stochastic ordering for internet congestion control and its applications. In: IEEE INFOCOM 2007–26th IEEE International Conference on Computer Communications, pp. 910–918 (2007). https://doi.org/10.1109/INFCOM.2007.111
8. Callegari, C., Giordano, S., Pagano, M., Pepe, T.: Behavior analysis of TCP Linux variants. Comput. Netw. **56**(1), 462–476 (2012). https://doi.org/10.1016/j.comnet.2011.10.002
9. Cardwell, N., Cheng, Y., Gunn, C.S., Yeganeh, S.H., Jacobson, V.: BBR: congestion-based congestion control. Commun. ACM **60**(2), 58–66 (2017). https://doi.org/10.1145/3009824
10. Centenaro, M., Costa, C.E., Granelli, F., Sacchi, C., Vangelista, L.: A survey on technologies, standards and open challenges in satellite IoT. IEEE Commun. Surv. Tutorials **23**(3), 1693–1720 (2021). https://doi.org/10.1109/COMST.2021.3078433
11. Claypool, S., Chung, J., Claypool, M.: Comparison of TCP congestion control performance over a satellite network. In: Hohlfeld, O., Lutu, A., Levin, D. (eds.) PAM 2021. LNCS, vol. 12671, pp. 499–512. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72582-2_29
12. Dai, C.Q., Zhang, M., Li, C., Zhao, J., Chen, Q.: QoE-aware intelligent satellite constellation design in satellite internet of things. IEEE Internet Things J. **8**(6), 4855–4867 (2021). https://doi.org/10.1109/JIOT.2020.3030263
13. De Sanctis, M., Cianca, E., Araniti, G., Bisio, I., Prasad, R.: Satellite communications supporting internet of remote things. IEEE Internet Things J. **3**(1), 113–123 (2016). https://doi.org/10.1109/JIOT.2015.2487046
14. Freimann, A., Dierkes, M., Petermann, T., Liman, C., Kempf, F., Schilling, K.: ESTNeT: a discrete event simulator for space-terrestrial networks. CEAS Space J. **13**, 39–49 (2021). https://doi.org/10.1007/s12567-020-00316-6
15. Fu, J.: TCP cubic memo. https://gist.github.com/fuji246/cffb0e460c14956d7357b57ea6823100. Accessed 13 May 2023
16. Ha, S., Rhee, I.: Taming the elephants: new TCP slow start. Comput. Netw. **55**(9), 2092–2110 (2011). https://doi.org/10.1016/j.comnet.2011.01.014
17. Ha, S., Rhee, I., Xu, L.: Cubic: a new TCP-friendly high-speed TCP variant. SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (2008). https://doi.org/10.1145/1400097.1400105

18. Kua, J., Loke, S.W., Arora, C., Fernando, N., Ranaweera, C.: Internet of things in space: a review of opportunities and challenges from satellite-aided computing to digitally-enhanced space living. Sensors **21**(23) (2021). https://doi.org/10.3390/s21238117
19. Le, H.D., Pham, A.T.: TCP over satellite-to-unmanned aerial/ground vehicles laser links: Hybla or cubic? In: 2020 IEEE Region 10 Conference (TENCON), pp. 720–725 (2020). https://doi.org/10.1109/TENCON50793.2020.9293761
20. Levasseur, B., Claypool, M., Kinicki, R.: A TCP cubic implementation in NS-3. In: Proceedings of the 2014 Workshop on NS-3, WNS3 2014. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2630777.2630780
21. Mészáros, Levente, Varga, Andras, Kirsche, Michael: INET Framework. In: Virdis, Antonio, Kirsche, Michael (eds.) Recent Advances in Network Simulation. EICC, pp. 55–106. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12842-5_2
22. Nguyen, D.C., et al.: 6G internet of things: a comprehensive survey. IEEE Internet Things J. **9**(1), 359–383 (2022). https://doi.org/10.1109/JIOT.2021.3103320
23. Obata, H., Ishida, K., Takeuchi, S., Hanasaki, S.: TCP-Star: TCP congestion control method for satellite internet. IEICE Trans. Commun. **89**(6), 1766–1773 (2006)
24. Peters, B., Zhao, P., Chung, J.W., Claypool, M.: TCP HyStart performance over a satellite network. In: Proceedings of the 0x15 NetDev Conference, Virtual Conference (2021)
25. Pirovano, A., Garcia, F.: A new survey on improving TCP performances over geostationary satellite link. Netw. Commun. Technol. **2**(1), xxx (2013). https://doi.org/10.5539/nct.v2n1p1
26. Postel, J.: Transmission control protocol. Technical report (1981)
27. Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., Scheffenegger, R.: CUBIC for fast long-distance networks. RFC 8312, February 2018. https://doi.org/10.17487/RFC8312
28. Roseti, C., Kristiansen, E.: TCP Noordwijk: TCP-based transport optimized for web traffic in satellite networks. In: 26th International Communications Satellite Systems Conference (ICSSC) (2008)
29. Shang, W., Yu, Y., Droms, R., Zhang, L.: Challenges in IoT networking via TCP/IP architecture. NDN Project (2016)
30. Tsipis, A., Papamichail, A., Angelis, I., Koufoudakis, G., Tsoumanis, G., Oikonomou, K.: An alertness-adjustable cloud/fog IoT solution for timely environmental monitoring based on wildfire risk forecasting. Energies **13**(14) (2020). https://doi.org/10.3390/en13143693. https://www.mdpi.com/1996-1073/13/14/3693
31. Varga, A.: Using the OMNeT++ discrete event simulation system in education. IEEE Trans. Educ. **42**(4), 11 (1999). https://doi.org/10.1109/13.804564
32. Varga, A., Hornig, R.: An overview of the OMNeT++ simulation environment. In: ICST (2010). https://doi.org/10.4108/ICST.SIMUTOOLS2008.3027
33. Zhao, P., Peters, B., Chung, J., Claypool, M.: Competing TCP congestion control algorithms over a satellite network. In: 2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC), pp. 132–138, January 2022. https://doi.org/10.1109/CCNC49033.2022.9700541