Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A machine learning based Distributed Congestion Control Protocol for multi-hop wireless networks

Juan Pablo Astudillo León [a], Luis J. de la Cruz Llopis [b,*], Francisco J. Rico-Novella [b]

[a] School of Mathematical and Computational Sciences, Yachay Tech University, Urcuquí 100119, Ecuador
[b] Department of Network Engineering, Universitat Politècnica de Catalunya (UPC), Barcelona 08034, Spain

ARTICLE INFO

Keywords:
Wireless multi-hop networks
Congestion control
Machine learning
QoS
Traffic prioritization

ABSTRACT

The application areas of multi-hop wireless networks are expected to experience sustained growth in the next years. This growth will be further supported by the current possibility of providing low-cost communication capabilities to any device. One of the main issues to consider with this type of networks is congestion control, that is, avoiding an excessive volume of data traffic that could lead to a loss of performance. In this work, a distributed congestion control mechanism is proposed for generic multi-hop networks. Different categories of data traffic are taken into account, each of them with different quality of service requirements. The mechanism is based on machine learning techniques, specifically, the CatBoost algorithm that uses gradient boosting on decision trees. The obtained decision trees are used to predict whether the packets to be transmitted over the network will reach their destination on time or not. This prediction will be made based on the network load state, which will be quantified by means of two parameters: the utilization factor of the different transmission channels, and the occupancy of the buffers of the network nodes. To make the values of these parameters available to all nodes in the network, an appropriate dissemination protocol has also been designed. Besides, a method to assign different transmission priorities to each traffic category, based on the estimation of the network resources required at any time, has also been included. The complete system has been implemented and evaluated through simulations, which show the correct functionality and the improvements obtained in terms of packet delivery ratio, network transit time, and traffic differentiation.

1. Introduction

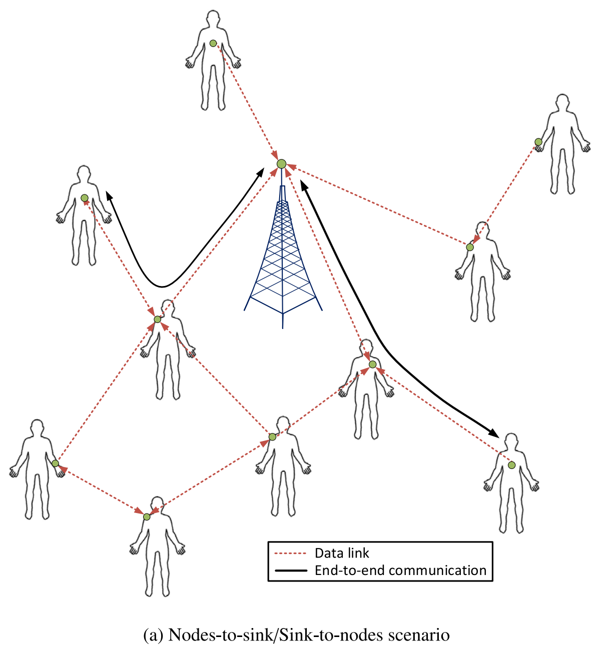
Wireless communications networks are currently one of the technologies that are attracting the attention of many researchers in the field of telecommunications. Among them, multi-hop wireless networks offer very suitable solutions in many application environments. In this type of networks, nodes can act both as source/destination of the data flows, and as relays of the data packets belonging to the rest of the nodes. On the other hand, depending on the purpose of the network, the nodes may remain static or may be mobile. In the latter case, there are a multitude of operational applications today, and there will undoubtedly be more in the future: ad hoc networks in emergency situations or protection missions [1], vehicular [2] and unmanned aerial vehicles communication networks [3–5], cell phones networks, networks of specific devices with a specific mission (e.g., robotic ants on a searching mission ...). It can be said that the applications of this type of networks will experience a constant increase in the following years, especially considering the increasing possibility of integrating communications capabilities in any device at a very low cost.

With this scenario in mind, it is also interesting to note that network traffic flows can be of different natures and generated by applications with different needs. For instance, in a specific environment, all the network nodes could generate traffic flows with the same destination (for instance, to a gateway connected to the Internet). The opposite situation is also possible (always the same source and different destinations), and of course, a combination of both, as shown in Fig. 1a. The most general situation would be a network in which all nodes generate/receive data traffic to/from the rest of the nodes (Fig. 1b). In this second scenario, a node could also act as a gateway to another network or to the global Internet.

One of the main problems in this type of networks is the possibility of trying to overuse the transmission channels, which can lead to situations of network congestion. In these situations, the quality of service offered to the applications becomes lower than required. Therefore, a congestion control mechanism is needed to prevent the network from getting into these situations. Traditionally, congestion

* Corresponding author.
E-mail addresses: jastudillo@yachaytech.edu.ec (J.P. Astudillo León), luis.delacruz@upc.edu (L.J. de la Cruz Llopis), francisco.jose.rico@upc.edu (F.J. Rico-Novella).

https://doi.org/10.1016/j.comnet.2023.109813
Received 16 September 2022; Received in revised form 28 March 2023; Accepted 6 May 2023
Available online 12 May 2023



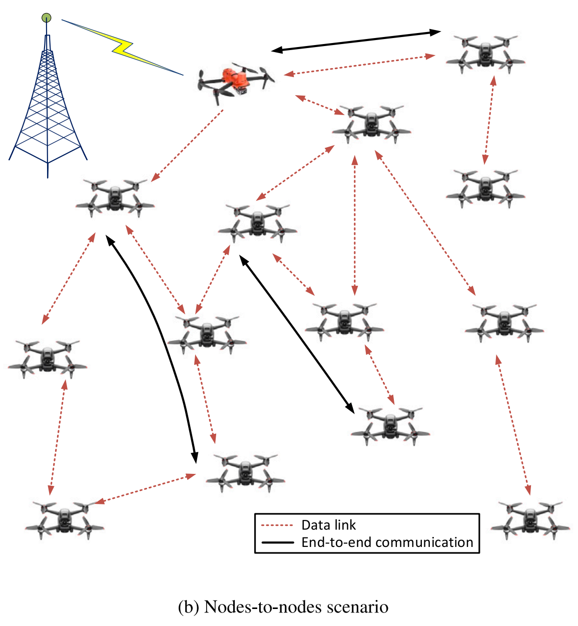


Fig. 1. Wireless multi-hop networks.

control techniques have been divided into two main groups. In the first group, the internal nodes of the network are responsible for detecting the possible congestion situation, by monitoring the occupation of the buffers associated with the transmission channels. When this occupation is high, the nodes generate explicit congestion notification (ECN) messages, which are sent to the endpoints of the communication. When these messages are received, the endpoints must activate the necessary functions to regulate the transmission rate. In the second group, no action is taken by the network nodes, and the responsibility for congestion control is left to the communication endpoints. This is the solution most widely adopted by TCP/IP networks. Indeed, depending on the transport protocol used, the approaches to obtain the congestion control are different. For example, when using a reliable control protocol (TCP alike), the proposed solutions are generally based on the control of the size of the congestion window. On the other hand, if the transport protocol is unreliable (UDP alike), other mechanisms must be used since there are no transport-layer connections between the source and the destination processes of the data packet flows.

In this paper, a congestion control mechanism is proposed for applications using an unreliable transport protocol. The mechanism is based on machine learning (ML) techniques, namely the use of decision trees. The use of ML techniques has been proven useful in many issues related to communication networks [6–10]. The proposed mechanism performs its predictions as a function of the network load state. For this purpose, a protocol that disseminates, among all the network nodes, the variables that quantify the network load state, has also been designed and implemented. The first objective is to guarantee that the transmitted data packets will arrive successfully at their destination. It is not only taken into account that the packets must not be lost on their way to their destination, but also that they must reach their destination on time, i.e., with a delay lower than a maximum bound. This bound depend on the time requirements of the applications that generate the packets. In addition, it is considered that different applications using the network may have different time requirements. On the other hand, the relevance of these applications may also be different (some of them may be critical for different reasons, while others may offer services of lesser importance). Therefore, in this proposal, the applications are grouped into different categories, being possible to assign different priorities to each category.

The rest of this article is organized as follows. Some related work is presented in the next section. Section 3 summarizes the general design and operation of the proposed mechanism. Section 4 presents the details of the machine learning techniques used. In this work, we have selected the use of an advanced variation of decision trees. The proposed strategy to generate the necessary datasets to train these decision trees, and their performance evaluation, are also included. Section 5 presents the designed protocol for the dissemination of the data (features) used as inputs by the decision trees in every node, and for the operation of the congestion control mechanism. The verification of the correct operation of the protocol and its performance evaluation has been carried out by means of simulations. The results are shown in Section 6. Finally, Section 7 presents the conclusions of this work.

2. Related work

Nowadays, many research groups focus their work on improving the performance of multi-hop wireless networks. If the network nodes building are mobile, some fundamental issues to take into account are packet routing and congestion control. Depending on the degree of mobility of the nodes and the applications that will transmit/receive messages over the network, the solutions to be considered are different. Looking at the routing of data packets, there are multiple proposals, among which the use of the Optimized Link State Protocol (OLSR) [11,12] is one of the most widely accepted, given its particular ability to reduce routing control traffic. Another interesting approach is B.A.T.M.A.N. (Better Approach To Mobile Ad-hoc Networking), a project whose development can be followed at [13]. A proposal specially designed for vehicular ad-hoc networks (VANETs) can be found in [14], where authors present a new routing protocol which try to improve the decision of the next forwarding node based on four routing metrics: distance to destination, vehicles density, vehicles trajectory and available bandwidth.

On the other hand, congestion control involves a set of techniques to detect and correct a possible overuse of the network resources, which leads to performance degradation. Basically, an unregulated traffic generation rate by some source nodes, or even their geographical position or the network size, can give rise to a partial or complete congestion situation. The applied techniques differ depending on the type of transport protocol used. When using a connection-oriented protocol, such as Transmission Control Protocol (TCP), proposals are generally oriented towards congestion window size management [15]. However, this possibility is not available for applications that use a non-connection-oriented protocol, like the User Datagram Protocol (UDP), and therefore other mechanisms must be employed.

In [16,17], some proposals are presented and applied to wireless multi-hop networks with static nodes. The first of them is implemented on the intermediate nodes and considers the possibility of being in emergency situations. The second one delves into the need to offer an equitable distribution of network resources among all traffic generating nodes. Authors compare their proposal with the Enhanced Distributed Channel Access (EDCA) mechanism. EDCA is a link layer access control mechanism that provides transmission priority to higher access categories. However, as indicated in the article, the EDCA mechanism has limitations, such as inequity in network resource allocation. Although EDCA uses a scheme based on the priority of traffic classes, it may result in high-priority traffic consuming most of the available bandwidth, depriving low-priority traffic of access to the wireless medium. In addition, EDCA does not guarantee minimum delay or maximum jitter for high-priority traffic. As another example, although not directly applied to multi-hop wireless networks, in [18], the authors present the mechanism used in WebRTC, where the Real-time Transport Protocol (RTP) over UDP is used. In this case, the rate control is performed using the information available in the receiver reports of the RTP Control Protocol (RTCP). One of the main limitations of WebRTC is that it requires a stable and reliable internet connection for proper functioning. It is highly dependent on network conditions, and low-bandwidth or unstable connections can result in poor audio and video quality or even dropped calls.

There are some congestion control schemes adapted to specific application environments, such as Data Center Transmission Control Protocol (DCTCP) [19,20], Data Center Quantized Congestion Notification (DCQCN) [21,22], SWIFT (Scalable and Stable Congestion Control with Rapid Fast Recovery) [23,24], and HOMA (High-speed On-chip Multiprocessor Architecture) [25]. DCTCP and DCQCN are designed for a data center environment, and require explicit network congestion notification (ECN) support, which may not be available in all scenarios. DCTCP estimates network congestion by calculating the fraction of packets marked with the ECN bit, while DCQCN assigns priorities to traffic classes and adjusts transmission rates based on congestion levels. On the other hand, SWIFT is based on the principles of delay-based and loss-based congestion control mechanisms. It uses a feedback loop to estimate the congestion level in the network and adjusts the transmission rate of the TCP flow accordingly. It requires modifications to the transport layer and NIC (Network Interface Card) firmware. With respect to HOMA, it uses a flow-based approach to provide low latency and high throughput while ensuring fairness among competing flows. It also requires explicit feedback from the network to allocate credits to each flow, which may introduce additional overhead and delay. Additionally, it assumes that all flows have the same level of congestion sensitivity. Therefore, DCTCP, DCQCN, SWIFT, and HOMA are very good congestion control mechanisms, but applied in specific environments, based on TCP and with the need for some kind of feedback, and so they are not applicable to control UDP traffic flows over generic wireless multihop networks.

Apart from the two most used transport protocols, TCP and UDP, there are other possibilities, such as QUIC, which is a UDP-Based Multiplexed and Secure Transport protocol, standardized in [26]. Some studies have shown that QUIC can reduce latency and increase throughput, particularly in scenarios with high packet loss rates and long round-trip times (RTTs) [27]. It focuses on flow-controlled streams for structured communication, low-latency connection establishment, and network path migration. It includes security measures that ensure confidentiality and integrity. Besides, a congestion control algorithm is also proposed in [28], but it is similar to the one used by TCP in TCP NewReno [29], and so it is also based in ECN, congestion window, and congestion control states (Slow Start, Congestion Avoidance and Recovery Period). Therefore, this is also a very well designed and promising protocol, but not applicable in the environment considered in this work.

The use of machine learning techniques is now widespread in many research fields. In the area of communication networks, relevant work has recently been proposed that improves the performance at the data link, network, transport, and application layers. Regarding wireless communication networks, numerous recent works can be found. A very good survey can be found in [30]. Furthermore, with respect to the congestion control issue, an excellent and recent review can be found in [31]. As can be read in this article, most of the works are mainly oriented again to the use of the TCP transport protocol.

For instance, authors in [32] present a proposal to be applied to the global Internet, combining classical congestion control techniques with new ones based on reinforcement learning. The proposal is called Orca and is deployed and evaluated over a global testbed on the Internet with servers located on five different continents. The evaluation shows the achievement of consistent high performance in different network conditions. In [33], a solution for cross-datacenter networks (multiple data center networks, DCNs, connected by a wide area network, WAN) is developed. Here, geographically distributed applications hosted on the cloud are taken into account. Delay signals are combined with explicit congestion notification (ECN) to form the proposed GEMINI strategy, which authors have implemented in the Linux kernel. The mechanism has been evaluated on a testbed, showing a low latency, high throughput, and fair and stable convergence.

In a previous work [34], we presented our first approach to congestion control based on machine learning techniques. The considered scenario was a Smart Grid Neighborhood Area Network. The main characteristic of these networks is that the nodes are static, and therefore the communication links between the network nodes and the network paths are also static and known a priori. Thus, the proposal was based on the characterization of all the data links of the network and their use to train the machine learning algorithms. In addition, communications (data flows) occur only between the network nodes and a special node with gateway functionality. In other words, the gateway is always either the source or the destination of the communication (nodes-to-sink/sink-to-nodes scenarios). On the other hand, the protocol for disseminating the necessary features was not implemented.

2.1. Contributions

The contributions of this work can be summarized as follows. First, we consider a much more generic scenario, where network nodes are mobile, and then the communication channels are not known a priori and variable over time. Second, the source and the destination of the data flows can be any pair of nodes in the network (nodes-to-nodes scenario). These generalizations required a redesign of the proposed mechanism and a new approach to training the ML algorithms. Besides, we have designed and implemented the complete protocol to disseminate the values of the selected features. In addition, an explicit mechanism to provide priority differentiation to the traffic categories has also been designed and included. These contributions collectively enhance the applicability, effectiveness, flexibility, and performance of the proposed congestion control mechanism.

3. General design and operation

The congestion control mechanism to be implemented has to satisfy several requirements. First, a central control unit must not be required, that is, the protocol must be distributed. Additionally, the mechanism should be able to operate on changing network topologies, since the network nodes are mobile. Moreover, the source and destination of the data flows can be any pair of network nodes. The source nodes must discard the transmission of a data packet if there is no guarantee

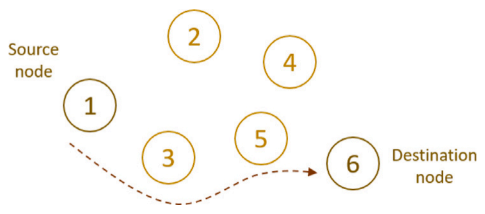


Fig. 2. Data packet path.

that it will reach its destination on time. This approach ensures that network resources are not wasted by transmitting data packets that will not be useful at their destination. Furthermore, the mechanism must be able to handle applications with different quality of service (QoS) requirements, particularly in terms of maximum network transit time (NTT). Finally, the mechanism must allow the assignment of different priorities to the applications, depending on how critical they are for the network or the users.

The mode of operation will be as follows. Each time a source node must transmit a data packet to a certain destination node, it must predict (by means of ML techniques) whether that packet will arrive on time (before a maximum NTT, which depends on the traffic category) at its destination. This prediction will be made based on the network load state at any time. The network load state will be quantified based on two parameters measured by each network node: the channel busy ratio (CBR) and the queued (buffered) packets (QP). Specifically, the source node is interested in knowing the value of these parameters at the nodes through which the data packet will travel. For instance, in Fig. 2, the source node (node 1) would be interested in the values at nodes 1, 3, 5 and 6.

In fact, different possibilities arise at this point, depending on the number of input features to be used in the prediction algorithm. That is, node 1 could use the parameters of all the nodes on the path, or it could reduce the number of parameters, using, for instance, only the values measured by the source and destination nodes (nodes 1 and 6 in the figure). It must also be noted that the source node does not know the complete list with all the intermediate nodes. It only knows which is the source node (itself), the destination node, and the second hop node (the node to which it must transmit the packet and which will be in charge of relaying it to its destination) provided by the routing algorithm. To find out which are the rest of the nodes in the path (node 5 in this example), additional control messages must be added to the protocol, which would increase the added overhead. In addition, given the dynamic nature of the networks under consideration, the intermediate nodes could change during packet hop-by-hop retransmissions, so that the algorithm would be using incorrect nodes. Therefore, in this proposal, we will consider only the cases in which the input parameters are those measured by the source node, second hop node, and destination node. In fact, in the training phase of the prediction algorithms, the resulting performance will be compared in three cases: (i) using only the source node, (ii) using source and destination nodes, and (iii) using source, second hop, and destination nodes. The details about the prediction algorithm are provided in Section 4.

On the other hand, as can be seen from the exposed mode of operation, all nodes must know the CBR and QP values measured by the rest of the nodes in the network. Therefore, all nodes in the network must periodically measure their own CBR and QP parameters and disseminate them to the rest of the nodes. The operation mode and the designed protocol are detailed in Section 5. As a result of the application of this protocol, the nodes will have the more recent values measured by the rest of the nodes, and will be able to apply them as input features in the prediction algorithm. Note that the application of the congestion control mechanism is only carried out by the source nodes. These nodes are the ones that predict if the total network transit time of each new data packet will be less than the maximum limit. If so, the packet is transmitted, otherwise it is discarded. Intermediate nodes in the network do not need to apply congestion control, since the source node has already done it. In fact, intermediate nodes have no information on how long the packet has been in the network since it was transmitted at its source, and therefore they could not apply the congestion control mechanism properly.

Finally, an optional second block is included in the mechanism to assign different transmission priorities to the traffic categories, depending on the relevance of each application (Section 5.4).

4. Machine learning based congestion control mechanism

4.1. Machine learning model design and operation

Predicting the state of a network is challenging, especially when we have mobile nodes in a wireless multi-hop scenario, due to several factors, such as the dynamic network topology, unpredictable interferences, limited bandwidths, and the lack of global knowledge, among others. Then, in this proposal, we are not trying to predict the state of the network. What we predict is, given a network state at the time of transmission of each data packet, whether this packet will arrive at its destination on time or not. That is, we know (to some extent) the network's state at the time of transmission of each packet (thanks to the disseminated CBR and QP values, which are current and real measures), and we predict, given this network state, whether the packet will arrive at its destination on time or not. These measures (CBR and QP values) are taken periodically by each network node, and disseminated to the rest of the nodes using, as will be explained in Section 5, the proposed DICCP protocol.

Supervised learning algorithms have shown promising results in wireless data transmission scenarios, such as vehicular communications and smart grids [8–10]. The accuracy of these algorithms relies on the quality and quantity of the labeled data used in the training phase. Thus, in this work, we propose a congestion control mechanism based on supervised learning. The general steps followed for the development and operation of the model are detailed in Fig. 3. As we are trying to keep the algorithm as simple as possible, to be executed in simple devices, we do not include all possible inputs as features for the learning algorithm (RSSI of neighboring devices, SNR, BER, MAC layer losses, …), because these parameters are reflected to some degree in the selected features (CBR and QP).

The chosen supervised machine learning algorithm is an advanced version of decision trees. Fig. 3 shows the workflow diagram used to build and operate this algorithm. The first stage contemplates the data collection scenario, where extensive simulations were conducted to collect representative samples, allowing the algorithm to learn from various traffic link and network state situations.

The obtained dataset is unstructured, so the second stage comprises the processing of the gathered data to extract the features and labels, which will be used later as inputs and outputs to train the ML models. Then, the third stage is the training phase, in which we will build and train the most suitable learning algorithm. In this phase, we have employed hyperparameter optimization techniques to efficiently select the best model configuration, seeking the highest possible level of accuracy. We have evaluated and validated the generated models by means of well-known ML metrics, such as the Receiver Operating Characteristic (ROC) curve. Finally, the last phase represents the operation of the congestion control mechanism, where the learning models are exported and operated in a realistic network.

In the following subsections, we will explain each of these phases in detail.

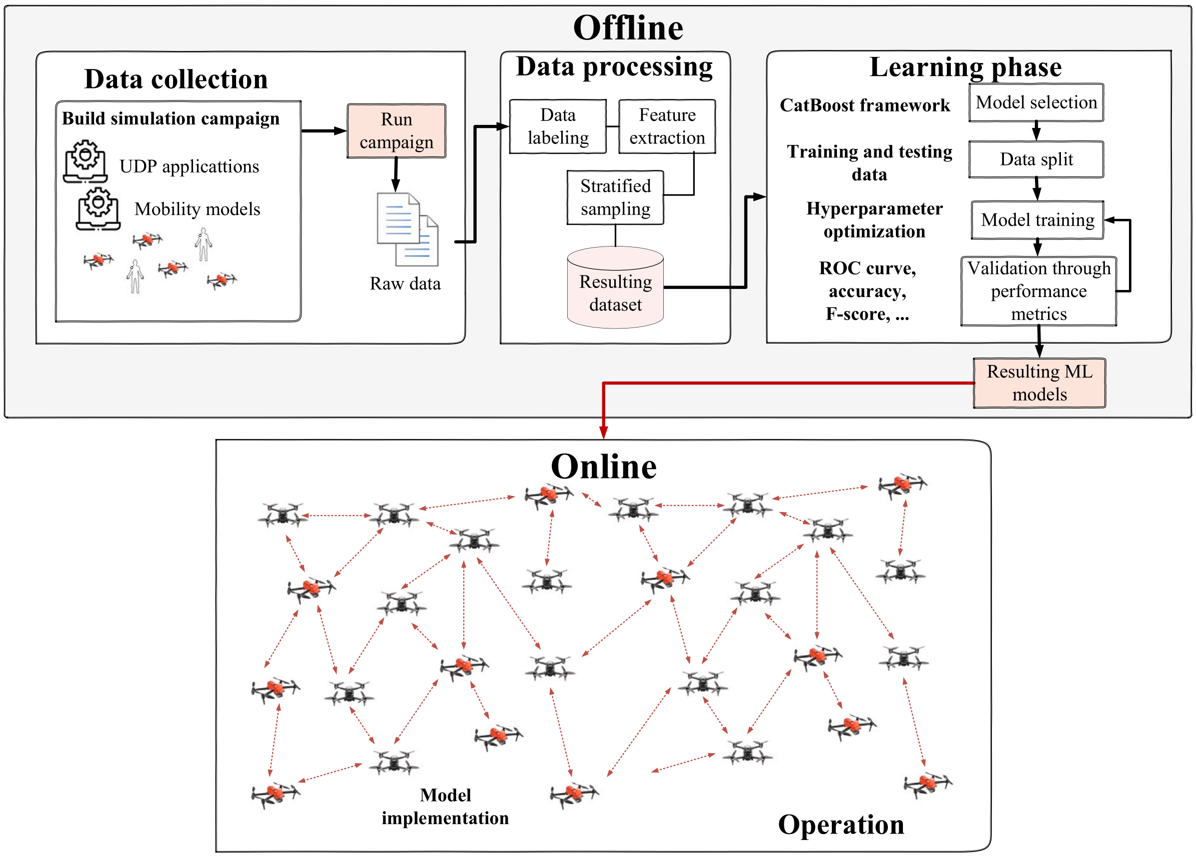


Fig. 3. Machine learning algorithm development and operation phases.

4.2. Data collection

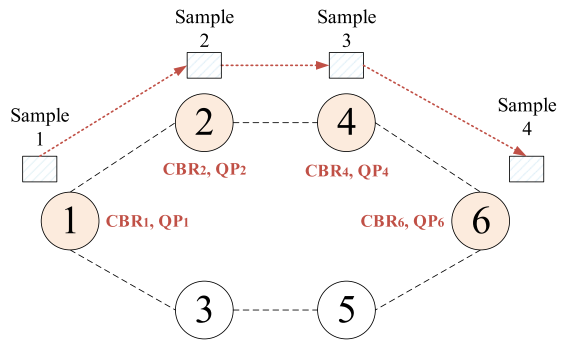
The basis of any machine learning-based algorithm is data. For our network scenario, general datasets are not useful, because we are working with specific features and a generic scenario, with traffic flows belonging to different categories and with different QoS requirements. Therefore, we have built a data collection scenario using the ns-3 simulator [35]. Fig. 4 illustrates a simple network topology to explain the data collection process. In this example, we have a network topology consisting of a source node, a destination node, and four relay nodes. In this case, a packet is transmitted from node 1 to node 6 through nodes 2 and 4, which is the best route to the destination calculated by the routing protocol. We have collected the network state information perceived by the current node, the second-hop node, and the destination node as the packet travels through each node in the network, along with the sampling time values. As can be seen, the different events associated with each packet transmitted or received during the simulation are captured.

To build the training dataset, we have conducted several sets of simulation runs, taking into account different traffic patterns for the UDP applications generated by each node. Each simulation consists of five hundred seconds of network operation, where different seeds of pseudorandom generators are used. Table 1 shows the configuration values used in the simulator to obtain the widest possible range of values for each variable measured during the data collection process. To prevent issues of underfitting or overfitting, the size of the datasets was determined by analyzing the learning curve. This curve illustrates the relationship between the accuracy of classification and the number of samples present in the dataset.

We have considered a network size of sixteen wireless mobile nodes which transmit using the IEEE 802.11ac wireless physical layer standard. The selected VHT-MCS (Very High Throughput Modulation and Coding Scheme) is VHT-MCS 0, with a single 20 MHz channel, and a single spatial stream, so that transmissions occur at a physical rate of 6.5 Mbps [36]. Moreover, for the UDP applications, an exponential distribution has been chosen for the packet size and for the packet inter-arrival time. Regarding the mobility model, a generic option has been chosen, the random waypoint mobility model, since it is a very generic model, with a node speed of 3.5 m/s. At this point, it is worth mentioning that, if the proposed mechanism is to be applied to nodes with a very different mobility model, to ensure the best performance the machine learning algorithms should be trained again. This training is done offline, prior to the network start-up, in order to make the trained algorithms available to each network node.

With regard to the queueing discipline, we have chosen the default ns-3 DropTailQueue, a simple queuing discipline that drops packets when the queue reaches its maximum capacity. In this queue, packets are transmitted on a First Come First Served (FCFS) basis. If the queue is full when a new packet arrives, it is dropped. While this basic queuing discipline can offer satisfactory performance in many situations, it lacks the capability to manage congestion or prioritize packets based on their importance. Therefore, a more complex queuing discipline could be needed. However, with our proposal, this need is avoided, because we provide fairness and/or priorities to the different traffic categories at the application layer.

For instance, the EDCA (Enhanced Distributed Channel Access) mechanism has been specifically designed to offer Quality of Service (QoS) support for wireless networks. This mechanism is able to provide QoS to four Access Categories (ACs), namely Voice (VO), Video (VI), Best Effort (BE), and Background (BK). However, this mechanism works at the data link layer, and in an independent way in each wireless hop. Therefore, it cannot guarantee a maximum network transit time (NTT), which is the most important issue for our proposal: being able to predict and avoid the transmission of packets that will not arrive on time at



Sample	CBR_{SOURCE}	QP_{SOURCE}	$CBR_{SECOND\ HOP}$	$QP_{SECOND\ HOP}$	$CBR_{DESTINATION}$	$QP_{DESTINATION}$	time
1	CBR_1	QP_1	CBR_2	QP_2	CBR_6	QP_6	t_1
2	CBR_2	QP_2	CBR_4	QP_4	CBR_6	QP_6	t_2
3	CBR_4	QP_4	CBR_6	QP_6	CBR_6	QP_6	t_3
4	CBR_6	QP_6	-	-	-	-	t_4

Fig. 4. Data collection.

Table 1
Main simulation parameters.

Description	Value
Network simulator.	ns-3.33
Simulation time	500 s
Transport layer	UDP
Random number generator	MRG32k3a
Network size	16
Mobility model	Random waypoint mobility model
Node speed	3.5 m/s
Packet length average (Bytes) and PDF	100 Exponential
Packet interarrival time and PDF	Exponential
Routing protocol	OLSR
Routing metric	Hop count
Queue discipline	first-in, first-out (FIFO)
Maximum queue size	500
Wireless physical layer	IEEE 802.11ac
Channel width	20 MHz
RxSensitivity: The energy of a received signal should be higher than this threshold (dBm) for the PHY to detect the signal	−101.0 dBm
CcaEdThreshold: The energy of a non Wi-Fi received signal should be higher than this threshold (dBm) to allow the PHY layer to declare CCA BUSY state.	−62.0 dBm
Transmission and reception gain (dB)	0 dB
Transmission and reception power (dBm)	1 dBm
Modulation Coding Scheme (MCS)	0
Short guard interval	0
Frame aggregation factor	0
Number of spatial streams	1

their destination. Thus, as we are resolving this issue at the application layer, also providing fairness and/or traffic prioritization (as desired or needed in the possible different application scenarios), we have considered that it is not necessary to activate the EDCA mechanism at the data link layer, keeping it simpler.

Finally, the paths between source and destination nodes are found using the OLSR protocol. It is worth mentioning that the proposed congestion control mechanism does not rely on a specific routing protocol, because the only need is to know which is the second hop node to the destination, and this information must be provided by any routing protocol.

4.3. Processing and feature extraction

As previously mentioned, the collected data are stored in an unstructured format. Therefore, we need to process and organize the variables of the collected data in a structured way. The main objective of this phase is to obtain the final dataset, which will be used to train the machine learning algorithms. We have processed the data to have a meaningful sample for each packet. Each processed sample represents the network state perceived by a source node before transmitting a packet to the destination, together with a label that represents whether this transmission has been successful or not (that is, if the data packet has reached its destination meeting the QoS requirements of the traffic category to which it belongs). For this purpose, we have defined four categories, each with a maximum allowed NTT. The first category is the one that encompasses time-critical applications, while category 4 has the lowest time requirements, as shown in Table 2. In this way, if the NTT is lower than the maximum allowed according to the respective category, the packet is considered to have been successfully received. Otherwise, the transmission is considered failed.

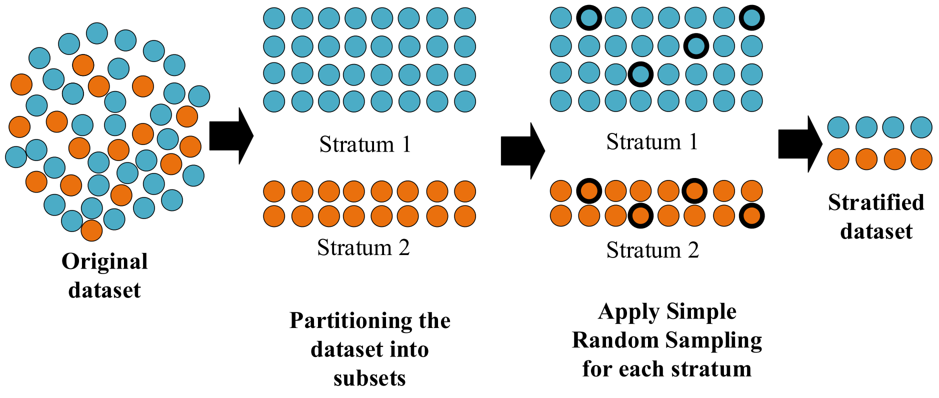
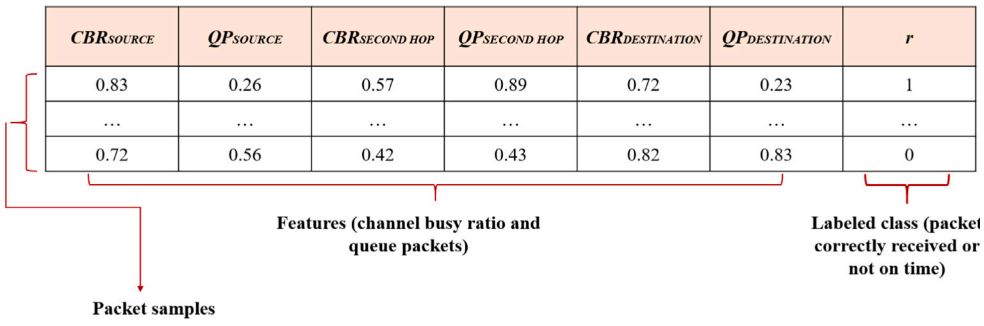

Original dataset
Stratum 1
Stratum 2
Partitioning the dataset into subsets
Stratum 1
Stratum 2
Apply Simple Random Sampling for each stratum
Stratified dataset

Fig. 5. Example of stratified sampling.



CBR_{SOURCE}	QP_{SOURCE}	$CBR_{SECOND\ HOP}$	$QP_{SECOND\ HOP}$	$CBR_{DESTINATION}$	$QP_{DESTINATION}$	r
0.83	0.26	0.57	0.89	0.72	0.23	1
…	…	…	…	…	…	…
0.72	0.56	0.42	0.43	0.82	0.83	0

Features (channel busy ratio and queue packets)
Labeled class (packet correctly received or not on time)
Packet samples

Fig. 6. Final format of the training and testing dataset.

Table 2
Maximum NTT allowed for each category.

Category	Network transit time
1	50 ms
2	200 ms
3	500 ms
4	1000 ms

Finally, before generating the final dataset, we have carried out an additional processing step to split the dataset into subsets, called stratified sampling [37]. The stratified sampling technique forces the distribution of the target variable(s) (i.e., the network transit time) among the different splits to be the same, or meet a specific condition. In this sense, we have divided the original dataset as a function of the NTT values, in 50 ms intervals. That is, we have a subset of sample data with a NTT value from 0 to 50 ms, another from 51 to 100 ms, and so on, where each subset has the same number of samples.

Fig. 5 shows an illustrative example of this process. In this example, we have a dataset that has two strata or sets of samples. On the one hand, a set of samples containing packets with an NTT from 0 to 50 ms (stratum 1), and on the other hand, the second set of data containing packets with an NTT from 51 to 100 ms (stratum 2). As can be seen, stratum 1 has more samples than stratum 2, and with the stratified sampling technique, we get the same amount of samples for each stratum. This small change results in an improved model prediction accuracy [38].

As a result of all the above operations, a dataset as shown in Fig. 6 is obtained, which contains the desired data to train the ML algorithms. Each training sample represents the parameters of a single packet from the point of view of the source node. The size of the datasets obtained in our experiments is around 46 GBytes before applying the stratification technique, and 2.4 GBytes after its application.

4.4. *Learning phase*

This section presents the ML algorithm chosen. In addition, we use the Exhaustive Grid Search method, which belongs to the set of hyperparameter optimization algorithms [39]. It allows a more efficient selection of the parameters of a learning model. Finally, the prediction models are evaluated using well-known ML metrics such as accuracy, ROC curve, F-score [40], and Cohen's kappa [41,42].

4.4.1. *Model selection*

The proposed mechanism corresponds to a binary classification problem, since the source node will decide between transmitting or discarding each packet depending on the current network load state. We have considered decision trees as a suitable option for this task. However, we have decided not to use simple decision trees, as they are susceptible to underfitting and overfitting problems [43]. For this reason, we have chosen the CatBoost algorithm [44], which uses the theory of decision trees and gradient boosting to combine several weak models sequentially into a robust competitive prediction model. CatBoost has been applied in multiple applications and provides promising results in recommendation systems, personal assistant, self-driving cars, weather prediction, and many other applications [44].

As previously said, different combinations of features can be used as inputs for the CatBoost algorithm (CBR and QP of the source node, destination node, etc.) We have compared the performance obtained with different combinations, as shown in Table 3. The first three versions consider only the CBR values of the source node, destination node, and second hop node, while the last three versions also consider the number of packets in the queue. In this way, we will be able to determine which combination is the most appropriate.

The operation of the CatBoost algorithm is shown in Algorithm 1. The full version of the algorithm can be found in [44]. The inputs to the algorithm are the training data X (features) and y (labeled class), as well as the hyperparameters p. The hyperparameters are the parameters

Table 3
Different versions considered for the ML algorithm.

Strategy	Features used to train the machine learning model
S_CBR	CBR_{SOURCE}
SD_CBR	CBR_{SOURCE}, $\text{CBR}_{DESTINATION}$
SDS_CBR	CBR_{SOURCE}, $\text{CBR}_{DESTINATION}$, $\text{CBR}_{SECONDHOP}$
S_CBR_S_QP	CBR_{SOURCE}, QP_{SOURCE}
SD_CBR_SD_QP	CBR_{SOURCE}, QP_{SOURCE}, $\text{CBR}_{DESTINATION}$, $\text{QP}_{DESTINATION}$
SDS_CBR_SDS_QP	CBR_{SOURCE}, QP_{SOURCE}, $\text{CBR}_{DESTINATION}$, $\text{QP}_{DESTINATION}$, $\text{CBR}_{SECONDHOP}$, $\text{QP}_{SECONDHOP}$

that are not updated during the learning process but are used to set up the model. For gradient boosting decision trees, the most important hyperparameters are the depth of the trees, the number of trees to combine in the boosting process, the learning rate, and the algorithm used to reduce the cost function. The result is a trained r model.

```
Algorithm 1: CatBoost Classifier
  Input: Training data $X$ (features) and $y$ (labeled class),
         hyperparameters $p$
  Output: Trained model $r$
1 Initialize model $r$ with hyperparameters $p$;
2 for iteration $i$ in 1 to $p.num\_trees$ do
3     Compute the gradient and hessian of the loss function for each
        training instance;
4     Train a new tree $t$ using the computed gradient and hessian;
5     Update the training loss using the trained tree $t$;
6     if the training loss has not improved for $k$ iterations then
7         break;
8     end
9     Add the new tree $t$ to the model $r$;
10 end
11 return $r$
```

Algorithm 1 initializes the model r with the specified hyperparameters p and then enters a loop that runs for $p.num\_trees$ iterations. In each iteration, the algorithm computes the gradient and hessian of the loss function for each training instance. The gradient and hessian are used to train a new decision tree t. The algorithm then updates the training loss using the trained tree t, which is added to the model r. Besides, we use the Early stopping technique [45–47], to get the most adequate value for the number of trees.

As can be seen, there are several hyperparameters to configure, and these can have a wide range of values. Therefore, we will use hyperparameter optimization to properly select the hyperparameter values, which is explained in detail in the next section.

4.4.2. Hyperparameter optimization

During the training process, a proper selection of the model hyperparameters is essential to get the most accurate predictions. Note that we have proposed different versions of the mechanism (i.e., a different number of features used in the training phase), and each version must deal with four different traffic categories (i.e., applications with different time requirements). For this reason, we have considered it essential to use hyperparameter optimization techniques, such as the Exhaustive Grid Search method [39]. Algorithm 2 presents this technique, where we have to provide a range of values for each hyperparameter that will be further evaluated. Table 4 shows the range of values for each hyperparameter and the percentage of data used in the training phase. As can be seen, Algorithm 1 (Algorithm 2, line 4) is executed with each combination of parameter grid values and is tested by calculating some performance metrics. Finally, Grid Search returns the hyperparameter configuration that showed the highest prediction score value.

To select the ranges of initial values provided for the hyperparameters, we have taken into account the following considerations. On the one hand, we try to get a good trade-off between model complexity and generalization capability using a tree depth range from 6 to 10, which is recommended by the authors of CatBoost [44]. Lower values do not provide adequate performance, while excessively high values may lead to overfitting.

```
Algorithm 2: Grid Search with CatBoost Classifier
  Input: Training data $X$ and $y$, hyperparameters grid $params$, $K$-fold
         cross-validation folds $f$
  Output: Best hyperparameters $\hat{p}$, best validation score $\hat{s}$
1 $\hat{p} \leftarrow \emptyset, \hat{s} \leftarrow -\infty$;
2 foreach $p$ in $params$ do
3     foreach $f_i$ in $folds$ do
4         Train $model \leftarrow CatBoostClassifier(p)$ using $(X_f, y_f)$;
5         Validate $model \leftarrow$ trained model on $X_{f_{val}}$;
6         $s \leftarrow$ Validation score of $model$;
7         if $s > \hat{s}$ then
8             $\hat{p} \leftarrow p, \hat{s} \leftarrow s$;
9         end
10    end
11 end
12 return $\hat{p}$, $\hat{s}$
```

Table 4
Training parameters and characteristics used for the hyperparameters optimization.

Parameter	Value
Training dataset	70%
Testing dataset	30%
Cross-validation	20
Maximum number of trees	3000
Loss function	Logloss
Validation-based	Early stopping
Number of CPUs used in parallel for training	80
Learning rate	[0.03, 0.1]
Tree depth	[6, 7, 8, 9, 10]

With respect to the number of trees, a high value can improve performance but also increase training time and memory requirements. As previously said, here we have used the Early stopping technique, which consists of repeating the training phase, using an increasing number of trees in each repetition, and stopping when no improvement in the accuracy is observed (Algorithm 1, line 6), avoiding overfitting. In order not to increase the number of trees indefinitely, a maximum value is taken. Here, we have chosen a value of 3000. This value is taken simply to have a high value so that it will not affect the number of trees finally chosen, but it will prevent the algorithm from running in an endless loop due to a bad configuration.

Fig. 7 shows the model configuration (i.e., the tree depth and the number of trees used to build the final tree model) and the testing accuracy of the resulting decision trees. As can be seen, the maximum value of 3000 for the number of trees is never reached in our experiments. We have generated twenty-four learning models based on gradient boosting decision trees thanks to hyperparameter optimization. There are six versions (see again Table 3), and each one includes four different traffic categories. As can be seen, the trees generated for category 1, which has the highest time requirements, are less complex, while the decision trees for categories 3 and 4, which are more permissive with respect to time, require a higher complexity of the model to have a high level of accuracy. Finally, it can be observed that as the number of features used for training increases, the model becomes more complex, and a higher predictive power is obtained, as shown in the accuracy plot.

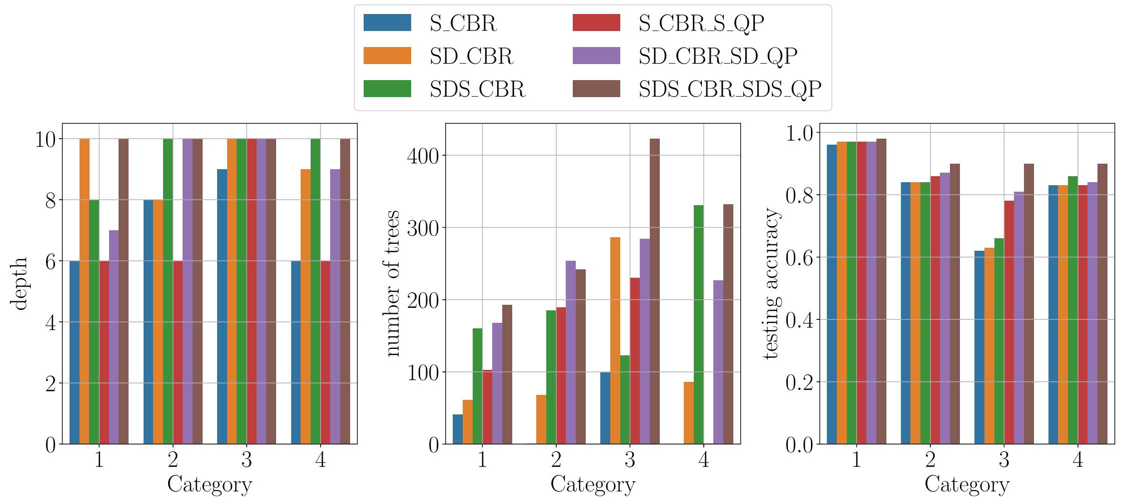


Fig. 7. Hyperparameter selection and accuracy.

Table 5
Decision trees performance metrics.

Strategy	F-score				Cohen's kappa			
	Category 1	Category 2	Category 3	Category 4	Category 1	Category 2	Category 3	Category 4
S_CBR	0.448	0.218	0.285	0.904	0.432	0.172	0.133	0.0
SD_CBR	0.560	0.229	0.402	0.905	0.546	0.184	0.181	0.014
SDS_CBR	0.475	0.245	0.579	0.916	0.461	0.200	0.299	0.458
S_CBR_S_QP	0.528	0.55	0.749	0.904	0.514	0.472	0.561	0.0
SD_CBR_SD_QP	0.658	0.591	0.781	0.908	0.645	0.517	0.619	0.153
SDS_CBR_SDS_QP	0.725	0.686	0.879	0.942	0.713	0.627	0.792	0.618

4.4.3. *Model validation*

In this section, the performance evaluation of the built models is presented. In a first evaluation, the predictive models have been assessed using the ROC curve and AUC (Area Under the Curve) metric. On the one hand, the ROC curve represents in a two-dimensional plot the true positive rate (TPR) on the y-axis versus the false positive rate (FPR) on the x-axis. The TPR (also called *sensitivity* or *recall*) represents the proportion of positive samples that were correctly classified as positive. The FPR (also called *fall-out*) represents the proportion of negative samples that were incorrectly classified as positive. Therefore, we would expect a good classifier to show the ROC curve as close to the upper left corner as possible (i.e., values close to 1 for TPR and close to 0 for FPR). Besides, the red diagonal in the ROC curves represents a random model which does not have any ability to distinguish between the two classes (i.e., the predicted probabilities of the two classes overlap), and therefore, TPR = FPR at any threshold. On the other hand, we have also used the Area Under the Curve (AUC), which numerically complements the result obtained by the ROC curve. The closer the AUC value is to 1, the more accurate the model predictions are.

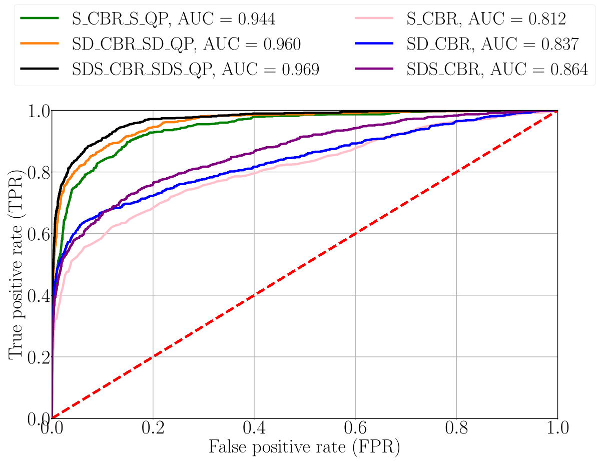
Fig. 8 shows the ROC curves and AUC values for the different versions of the prediction model and for the different traffic categories. Two main conclusions can be drawn from the graphs. First, as the number of features increases, the model predictions become more accurate. Second, the performances obtained are better when both types of available features (CBR and QP) are used, than when only one (CBR) is used. On the other hand, similar to what happens with the accuracy metric, applications with the lowest time requirements show the worst results, especially when few features are used for training. Therefore, from these first results, we can conclude that the designed decision tree-based classifiers show a high percentage of accurate predictions when using all CBR and QP features in the learning phase.

In a second evaluation, the ML models have also been assessed through F-Score and Cohen's kappa metrics. The F-Score metric represents the weighted harmonic mean of the precision and recall metrics, where the best score is 1, and the worst is 0. On the other hand, Cohen's kappa indicates the level of agreement between two different raters. Here, the best score is also 1 [48], and negative values are possible for inaccurate models. The results are presented in Table 5, showing again that better results are obtained when the CBR and QP features for the source, second hop, and destination node are taken into account in the training process. Furthermore, looking closely at the results, we have Cohen's kappa values of 0 for category 4. Specifically, this happens when only the features of the source node are used in the training phase. Notice that a Cohen's kappa score of 0 means random agreement among raters, whereas a score of 1 means a complete agreement between the raters. In other words, if the Cohen's kappa is 0, the observed concordance coincides with that which would occur by pure random chance. Positive values indicate greater agreement than would be expected by pure chance. If the result is 1, it would be a perfect match. Finally, the value for Cohen's kappa can be less than 0 (negative), meaning there is less agreement than random chance, a situation that does not occur in the presented models.

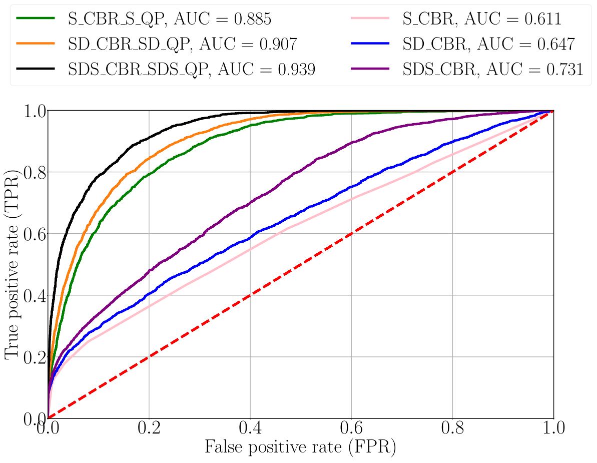
In this second evaluation, we can conclude again that the highest level of accurate predictions is achieved, for all versions and for all categories, when the CBR and QP features of the source, second hop, and destination node are used. Therefore, the *SDS_CBR_SDS_QP* strategy is the one that would be implemented and evaluated in the following sections.

4.5. *Operation phase*

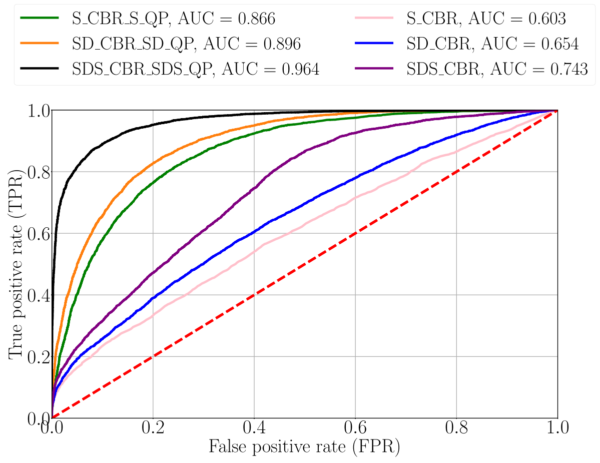
Once the model is suitably adjusted, it can be installed in the network nodes and put into operation. This way, each time a node has to transmit a new data packet generated by one of its applications, it can predict whether the packet will reach its destination on time or not, depending on the time requirements of each traffic category. The process to be followed is detailed in Algorithm 3. The node checks the traffic category to which the data packet belongs, and applies the



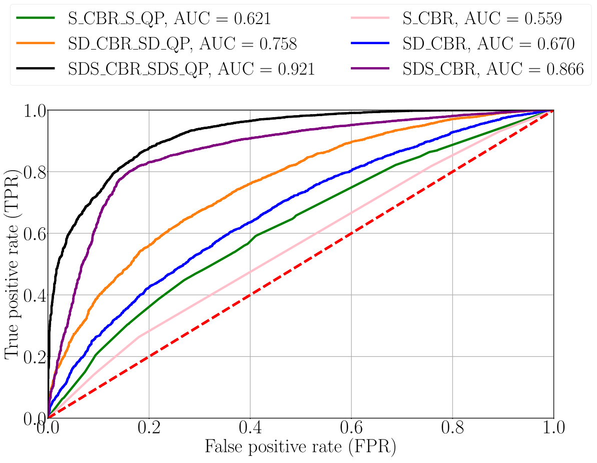
(a) Category 1



(b) Category 2



(c) Category 3



(d) Category 4

Fig. 8. ROC curves.

corresponding decision tree. As a result, it will obtain the prediction of whether the packet will arrive on time or not. Therefore, the packet will only be transmitted if it is predicted to arrive on time, otherwise it will be discarded. As an additional functionality, it can be taken into account if the traffic prioritization mechanism must be applied. This functionality is explained in detail in Section 5.4.

It is worth mentioning that recent developments in machine learning have been deployed with Python-based frameworks because of their flexibility, extensive library support, and active community of users. Nevertheless, sometimes the models must be ported to C or C++ code to run them on real low-cost devices or to be included in some network simulators, such as ns-3. Therefore, we have converted the models generated in Python through the CatBoost framework into readable C++ code for the network simulator.

Algorithm 3: Operation phase

```
Input: Supervised learning model, consisting of 4 CatBoost decision
       trees, one for each traffic category: $DT(k), k = 1..4$.
       Prioritization function.
Output: Transmission decision (Boolean)
1  Get the traffic category, $k$, of the data packet.
2  Get, from the data repository, the CBR and QP values needed as
     features for the decision tree.
3  Run the decision tree, $DT(k)$, obtaining the desired prediction
     (whether the packet will arrive to its destination on time or not)
4  if packet will arrive on time then
5      if traffic prioritization enabled then
6          Apply the prioritization function
7          if packet is prioritized then
8              Transmit packet
9          else
10             Discard packet
11         end
12     else
13         Transmit packet
14     end
15 else
16     Discard packet
17 end
```

5. Distributed Congestion Control Protocol

As previously explained, each network node performs periodic CBR and QP measurements. The measured values must be disseminated to the rest of the nodes so that they can be used by their congestion control algorithms. To perform this dissemination, we have developed the DIstributed Congestion Control Protocol (DICCP). As will be seen later, the messages of this protocol include the necessary fields for the dissemination of the CBR and QP values. Additionally, some other features have been added to the protocol that are also detailed in this section. The ideal transport mode for this type of protocols is unreliable transport, and so DICCP messages are encapsulated over UDP. Note that the proposed mechanism regulates the transmission of UDP flows corresponding to the different applications that use the

Table 6
DICCP parameters

Name	Description	Default value
DICCP_MIT	Interval time between DICCP messages	1 s.
DICCP_SIT	Interval time between CBR and QP samples	1 ms.
DICCP_CAT	Number of traffic categories	4
DICCP_EWMA_W	Weight applied to the new CBR and QP measures in the EWMA	0.005
DICCP_NThres	Threshold for the number of neighbors used by the forwarding algorithm	4
DICCP_NVT	Neighbors valid time	5 s
DICCP_ADP_PRI	Priority differentiation	True
DICCP_PRI_Factor	Priority Intensity Factor	1

Table 7
Network state (NS) table.

Name	Size [octets]	Description
NID	4	Node Id (IP Address)
CBR	2	Channel Busy Ratio
QP	2	Queued Packets
TS	8	Timestamp
RCAEst	2 · DICCP_CAT	RCA Estimation (optional)

Table 8
Neighborhood (NBH) table.

Name	Size [octets]	Description
NID	4	Node Id (IP Address)
TS	8	Timestamp

Table 9
RCA estimation (RCAEst) table.

Name	Size [octets]	Description
TC	1	Traffic Category
LAT	8	Last Arrival Time
RCAEst	2	RCA Estimated value

network. However, the UDP flow corresponding to the DICCP protocol itself will not be regulated, that is, its messages are always transmitted. This is not a problem with regard to the total load on the network, since the number of DICCP messages is very small and negligible compared to the rest of the flows.

5.1. DICCP parameters and repository tables

This section introduces, for reference, the protocol parameters, as well as the tables that each node must maintain for its operation. Table 6 summarizes the configurable parameters of the protocol, together with their default values. They will be explained in the following sub-sections.

On the other hand, the values necessary for the operation of the protocol are stored in three repository tables: the Network State (NS) Table, the Neighborhood (NBH) Table, and the Required Channel Availability Estimation (RCAEst) Table. Their contents are shown in Tables 7–9 respectively. The NS Table stores the CBR and QP values received from the rest of the nodes in the network, together with a timestamp that specifies the time at which these values were measured. As discussed, these values will be used as input features for the machine learning algorithms used for congestion control. As an additional possibility, which will be explained in detail later, the Required Channel Availability of each network node, and for each traffic category, is also stored. These values represent an estimation of the proportion of channel time that each node would require for each traffic category. They will be used, if needed, by the proposed prioritization algorithm. All the values in the NS Table are received, as will be seen in the next section, in the DICCP messages. In the NBH Table, each node stores the identifier of each of its neighbors (nodes from which it receives direct transmissions), along with the time instant at which the last transmission was received from each of them. The contents of this table will be useful both in the process of retransmission of the protocol messages and in the application of the proposed priority differentiation algorithm. Finally, in the RCAEst Table, it is stored, for each traffic category, the value of the arrival time of the last data packet, together with an estimation of the necessary channel availability for that traffic (the same information that, for the rest of the network nodes, is stored in the NS table, as mentioned above). This information will be also useful for the application of the priority differentiation algorithm, as will be later explained.

The responsibility for maintaining this repository tables belongs to the application layer, as the proposed mechanism works at this layer, and the values included in the tables are a consequence of the DICCP application protocol itself, and of the applications running on the node. That is, the content of the NSTable table is maintained at the application layer thanks to the values of the rest of the nodes (CBR, QP, . . .) received in the DICCP messages, as will be detailed in the next section. The content of the NBH table is also a direct consequence of the reception of DICCP messages generated by the neighbor nodes (the transmitter node identifier is also in the DICCP message). And finally, the content of the RCAEst table consists of values also generated and maintained by the application layer, because they are directly related to the applications running on the node. Additionally, when a node generates a DICCP message, it must include its own CBR and QP values, values that are available at the data link level, so for this functionality, a cross-layer communication between the application and link layers is necessary.

Regarding the memory size of the tables, the NS table has an entry for each network node, and each entry needs a maximum of 24 bytes (considering 4 traffic categories), as can be seen in Table 7. The NBH table has one entry per neighbor node, and each entry needs 12 bytes (Table 8). Finally, the RCAEst table has an entry for every traffic category (Table 9). Thus, if we consider a network with M nodes and with NN neighbors per node, the memory size needed for is 24 ·M bytes for the NS table, 11 ·4 bytes for the RCAEst table and 12 ·NN bytes for the NBH table. In general, and for a network size that is not excessively large, these memory needs should be assumed without problems by the network nodes. Otherwise, if they have a low amount of available memory, a reduction in the memory needs can be contemplated, decreasing the resolution of the timestamps, using 32 bits instead of 64. This will provide a valid resolution in the vast majority of the cases. Exceptionally, if the memory availability is extremely low, a maximum size for each table must be defined and configured. In this case, it is possible that some necessary values are not available to the nodes. Then, estimated values will be used by the algorithms, with the consequent decrease in performance.

5.2. Message format

The messages of this protocol carry the information shown in Fig. 9. The first 4 bits are used to specify the protocol version (PV, v1 currently). The next 4 bits indicate the message type (MT). Although only one message type is currently defined (Node Data Message, ND_MESSAGE), new types are planned to be added as new functionalities are incorporated in future versions of the protocol. The identifier of the node that generated the message is included below (Source Node

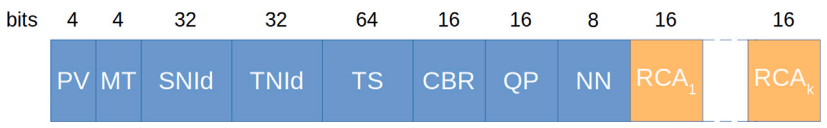


Fig. 9. DICCP message format.

Id, SNId), and then the identifier of the node which has (re-)transmitted the message (Transmitter Node Id, TNId). As we are working with IP networks, we use the IP addresses as node identifiers. Next, a timestamp (TS) corresponding to the time at which the message was sent by the generating node, followed by the CBR and QP values. To get these values, the nodes take samples every DICCP_SIT seconds (Table 6). To avoid large oscillations of these values, and to give importance to the previous states of the network, the current values are averaged with the previous ones by means of an exponentially weighted moving average. To do this average, the weight DICCP_EWMA_W (Table 6) is used. The next field corresponds to the number of neighbors of the transmitter node. This value will be used by the forwarding algorithm, as will be explained in the next section. The last fields are optional, and correspond to the estimation made by the nodes of the channel availability required for each traffic category. Their usefulness will be detailed later when discussing the possibility of assigning different priorities to the different categories.

5.3. *Message processing and forwarding*

Each network node generates and broadcasts a DICCP message every DICCP_MIT seconds (see Table 6). The value of this parameter must be adjusted according to the level of mobility of the network nodes. If the mobility is very high, the time between DICCP messages must be reduced, which has the disadvantage of a higher overload. On the other hand, if the mobility of the nodes is low, the time between messages can be longer, reducing the overload. To avoid node synchronization, a random jitter is added to the DICCP_MIT value. The receiver nodes must process and possibly retransmit the messages, following Algorithm 4. As it can be seen, the main mission of the message processing is to update the NS and NBH tables.

Next, the message must be broadcast again by the receiver, to disseminate the carried information to all the network nodes. Therefore, the message would be retransmitted by all the neighbors of the transmitting node, which may result in an excessive number of broadcasts. As a practical solution to reduce the number of broadcasts, the strategy adopted is that the nodes will retransmit the message with a certain forwarding probability, P_F, which will depend on the number of neighbors of the transmitting node (this is the reason why this number is included in the DICCP Message), following the expression in Eq. (1). As can be seen, the forwarding probability will be equal to 1 if the number of neighbors is below a certain threshold (DICCP_NThres, see Table 6). Otherwise, if the threshold is exceeded, the forwarding probability will be inversely proportional to the number of neighbors.

$$P_F = \begin{cases} 1 & NN \le DICCP\_NThres \\ \frac{DICCP\_NThres}{NN} & NN > DICCP\_NThres \end{cases} \tag{1}$$

The last step in the algorithm consists of updating the NBH Table. As previously introduced, in this table each node stores the identifier of each of its neighbors, together with a timestamp indicating the time instant at which the last transmission was received from each one. During the update, the node first queries its table to see if the node transmitting the message is already included. In this case, the timestamp is updated with the value of the current time instant. If the node was not included, a new entry is created. The remaining entries in the table (the rest of the neighbors) are then checked for validity. It must be taken into account that, as the nodes are mobile, existing neighbors can move out of the coverage range. This is why the time at which the last transmission was received from each neighbor is stored in the table. If the difference between the current instant and the timestamp stored for a particular node is greater than the value DICCP_NVT (see Table 6), it is considered that no more direct transmissions are received from that neighbor and consequently it is removed from the table.

Algorithm 4: DICCP message processing

```
Inputs : Message Type, $MT$; Source Node Identifier, $SNId$;
         Transmitter Node Identifier, $TNId$; Timestamp, $TS$;
         Channel Busy Ratio, $CBR$; Queued Packets, $QP$; Required
         Channel Availability Estimation, RCAEst; Neighbours
         Number, $NN$; Receiver Node Identifier, $RNId$; Network
         State Table, $NS\_Table$.
Result: DICCP Message processed (NS_Table and NBH_Table
        updated); Exit Code, $EC$.
1  if $MT \neq ND\_MESSAGE$ then
2      Discard message //The Message Type is not defined.
3      Exit ($EC = 1$ : MT not defined)
4  end
5  if $SNId = RNId$ then
6      Discard message //The message was generated by the receiver
         node
7      Exit ($EC = 2$ : Self-generated message)
8  end
9  if Entry in NS_Table for node SNId exists then
10     NS_TS ← TS from NS_Table
11     if $TS > NS\_TS$ then
12         Update NS_Table entry with the received values
             (CBR,QP,TS,RCAEst)
13         $forwardMessage(NN)$
14     else
15         Discard message
16     end
17 else
18     Create new entry in NS_Table with the received values (CBR, QP
         ,TS, RCAEst)
19     $forwardMessage(NN)$
20 end
21 $updateNBH\_Table(TNId)$
22 Exit ($EC = 0$ : Success)
```

5.4. *Quality of service and priority*

As mentioned above, the proposed congestion control mechanism is based on the prediction of the probability that application data packets reach their destination with a delay (NTT) lower than the maximum allowed. The machine learning algorithms detailed in the previous section are used to obtain this probability. When the probability of success (i.e., the packet arrives on time at the destination) is below a certain threshold, the packet will be discarded at source, avoiding the unnecessary waste of network resources. Besides, we have considered the coexistence of different data packet flows (traffic categories) with different QoS requirements, that is, different maximum allowed NTTs. Therefore, each time a node wishes to transmit a packet of a certain category k, it applies the decision tree corresponding to that category and decides whether to transmit or discard the packet. To do so, it will use as input features the values available in the NS Table, detailed in the previous sections.

On the other hand, the proposed protocol also provides the possibility of assigning different priorities to the different traffic categories, depending on the relevance of the applications that generate them. In this way, the packets that have passed the previous phase, must go through a new priority differentiation algorithm. For this new functionality, the proposed strategy is based on the knowledge of the proportion of time the channel needs to be available for the transmission of the packets of each category k (Global Required Channel Availability for category k, $GRCA_k$). The packets belonging to a given category will be transmitted with a certain probability, which will depend on the proportion of time to be reserved for the transmission of the packets belonging to higher categories. That is, if there are a total of K categories (the value of DICCP_CAT in Table 6), the transmission probability for packets with priority k (considering $k = 1$ the higher priority), P_{Tk}, will be:

$$P_{Tk} = \begin{cases} 1 & k = 1 \\ 1 - \sum_{i=1}^{k-1} GRCA_i & 1 < k \leq K \end{cases} \tag{2}$$

The problem now is how the network nodes can compute their own $GRCA_k$ values, taking into account that these values are variable over time, since they depend on the traffic generated at each moment by the running applications. To this end, each network node, depending on the data packets it has to transmit generated by the different applications, makes an estimation of the ratio of time it would need to use the channel to transmit them (an estimation of the node RCA for each category k, $\widehat{RCA}_k$). This ratio can be ideally computed as the ratio between the average time required for the transmission of a data packet and the average time between packet arrivals. Thus, each time a new packet becomes available for transmission, a new $\widehat{RCA}_k$ sample is calculated and averaged with the previous values by means of an EWMA using the DICCP_EWMA_W weight. These averaged estimated values are stored in the RCAEst Table (Table 9).

In addition, it must be noted that each node shares the channel with all its neighbors. At this point, we have considered two possible operation modes. The first and simpler mode would be applicable in scenarios in which all nodes transmit the same amount of traffic. In this case, the final $GRCA_k$ value would be NN (Neighbors Number) times $\widehat{RCA}_k$. The value of NN for each node is nothing more than the number of entries in its NBH Table.

Besides, in order to provide more flexibility to the priority assignment mechanism, reserving more or less channel resources for the higher priority traffics, we have included a priority intensity factor (DICCP_PRI_Factor, see Table 6) in the protocol. Its influence will be discussed in detail in the results section. Thus, the $GRCA_k$ values, $k = 1..K$, are finally obtained as follows:

$$GRCA_k = DICCP\_PRI\_Factor \cdot NN \cdot \widehat{RCA}_k \tag{3}$$

The second mode of operation is applicable in network environments where the different network nodes do not transmit the same amount of traffic. Now, in order to calculate the $GRCA_k$ value, each node needs to know the RCA estimations of its neighboring nodes. To this end, the nodes include their estimates in their generated DICCP messages, using the optional fields (Fig. 9). Thus, the nodes have the necessary information for the calculation (storing the received values in the NS_Table) at the expense of a higher channel utilization (larger packet size) by the control protocol. In this case, the $GRCA_k$ values, $k = 1..K$, computed by node n are:

$$GRCA_k^n = DICCP\_PRI\_Factor \sum_{m \in NS_n} \widehat{RCA}_k^m \tag{4}$$

where NS_n represents the set of neighbors of node n. Obviously, this second mode also operates correctly in the case of networks with nodes generating the same amount of traffic, but we have considered interesting the possibility of reducing the channel utilization due to control traffic in simple scenarios, as well as the size of NS_Tables.

Table 10
Application data packets

Parameter	Value
Packet generation rate (mean value)	0.5(Medium), 0.75(High) packets/s
Interarrival time distribution	Exponential
Packet Size (mean value)	800 octets
Packet Size distribution	Truncated exponential Max: 1500 octets Min: 46 octets

6. Simulation results

6.1. Scenario description

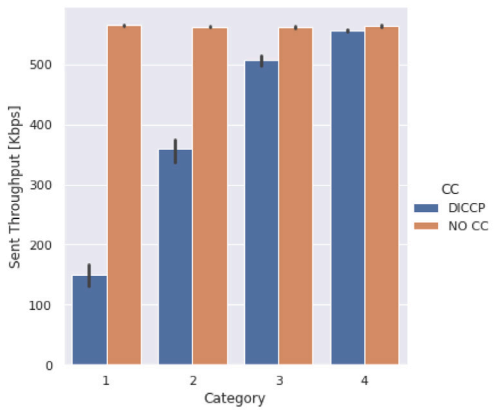
The basic scenario considered for the performance evaluation of the proposed algorithms and protocols consists of a rectangular area of 100 x 100 m. Within this area, 16 wireless stations are randomly placed. Each of these stations generates four traffic flows (belonging to four different categories) with destination in each of the other stations. That is, each station generates 4x15=60 traffic flows, which means a total of 16x60=960 traffic flows in the network. The packet generation rate (for every traffic flow) and size are detailed in Table 10. As we are interested in evaluating the different treatments that the proposed mechanism offers to each traffic category, an identical network load has been considered for each of them (i.e., equal values for both generation rate and packet size). For low-load network states, the congestion control mechanism has practically no impact in the nodes behavior (as expected), and therefore no performance differences are observed. Therefore, the results presented in this section have been obtained for medium and high load states. The rest of the relevant parameters of the simulation scenario are selected in the same way as previously done in the training phase of the machine learning model (see Table 1). Regarding the buffer size of the nodes, we have chosen the default value (500) in ns-3. This is a relatively large value, which allows the network to absorb more traffic. In a real network implementation, this value will depend on the type of node being used. The smaller this value is, the greater the probability of reaching a congestion situation, and therefore the need for the congestion control algorithm. To avoid a greater influence on the results presented, it has been decided to leave the default value, although it is worth highlighting the fact that the smaller the buffer size, the more relevant the improvements introduced by the proposed mechanism will be. In the same way, the QoS requirements considered for each of the traffic categories are the same as those previously presented in Table 2.

6.2. Congestion control

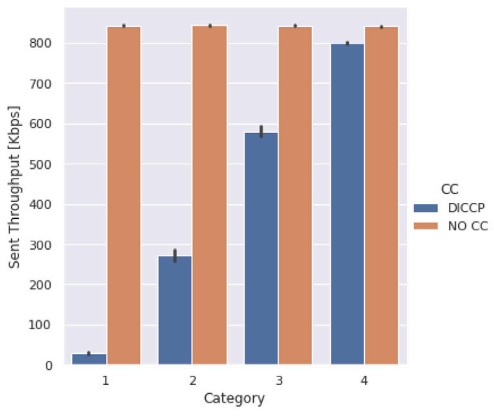
First, the results obtained when only the mechanism for congestion control is applied will be presented. In other words, each time a node generates a new packet to be transmitted, a decision is made as to whether to transmit it, based on the value returned by the decision tree corresponding to the category to which the packet belongs (see Algorithm 3). However, in this first set of results, the proposed differentiation of traffic based on its priority is not applied.

Since, according to the performance analysis performed in Section 4.4.3, the best performing strategy is the one that uses as features both the CBR and the QP of the source, second hop and destination nodes, this strategy is the one used for the evaluations performed in this section.

Fig. 10 shows the total data flow sent by all nodes in the network (Sent Throughput), separated by categories. The results obtained by applying the congestion control mechanism ("DICCP") are compared with those obtained without its application ("NO CC"). Looking first at the medium load case (Fig. 10(a)), we observe that, when no congestion control is applied, the same throughput is transmitted for all categories.

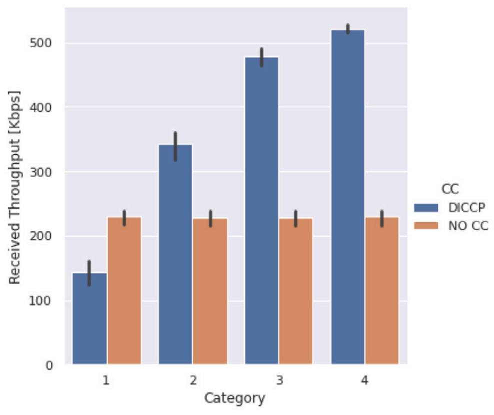


(a) Medium Load

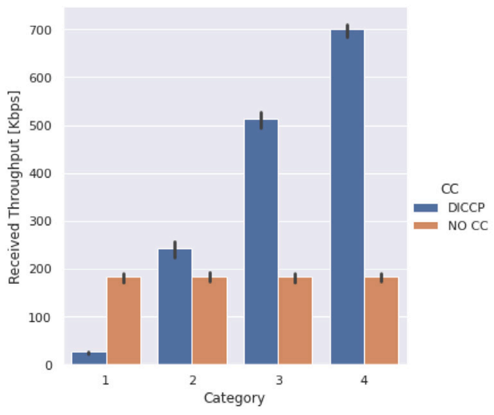


(b) High Load

Fig. 10. Sent throughput [Kbps].



(a) Medium Load



(b) High Load

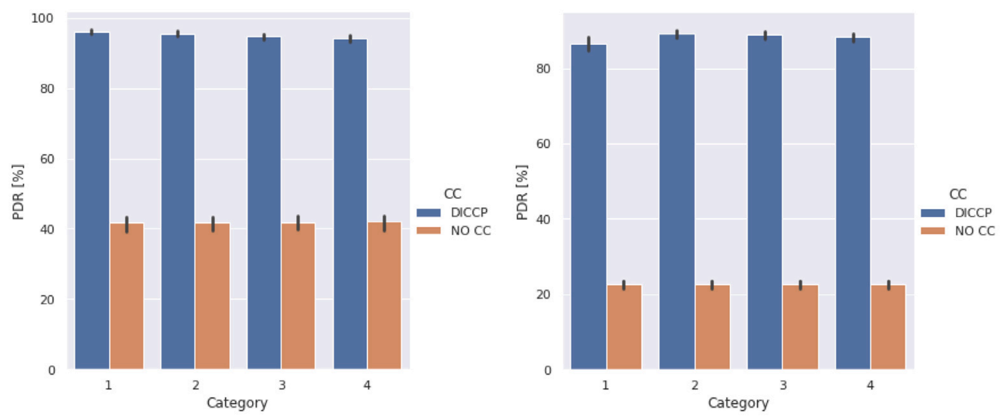
Fig. 11. Received throughput [Kbps].

However, when congestion control is applied, less throughput is transmitted for the categories with more stringent NTT requirements. This is the expected behavior, since the congestion control only enables the transmission of a packet when it is predicted that the packet will reach its destination meeting the NTT requirement of the category to which it belongs. Also, as expected, if the network load is higher (Fig. 10(b)), the transmission of categories with more stringent requirements is further slowed down. Thus, the first conclusion we can draw is that the decision trees have been well trained and are performing their mission correctly.

On the other hand, Fig. 11 shows the received throughput, i.e., the data flow that finally reaches its destination in the network. Looking again at the medium load case first (Fig. 11a), it can be seen that, although more throughput was sent without congestion control, more throughput is received, for most categories, when congestion control is applied. This difference is even greater when the network load is higher (Fig. 11b). The relationship between the received and sent throughput, i.e., the packet delivery ratio (PDR), is shown in Fig. 12, where it can be seen that this ratio is always higher when DICCP is applied. The improvements obtained are evidently more relevant when the network load is higher.

The next step consists of evaluating the "quality" of the received throughput. As mentioned above, each traffic category has a maximum value for the NTT, so packets received after this time are invalid and must be discarded by the receiver. Fig. 13 shows the average value of this time, separated again by traffic category. As can be seen, the application of DICCP results in a reduction of the NTT, which becomes more relevant again as the network load becomes higher. Finally, Fig. 14 shows the received throughput, but now taking into account only those packets that have reached their destination in compliance with the time requirements of their category (Compliant Received Throughput). As can be seen, the application of the proposed congestion control mechanism results in a significant increase in this throughput for all traffic categories.

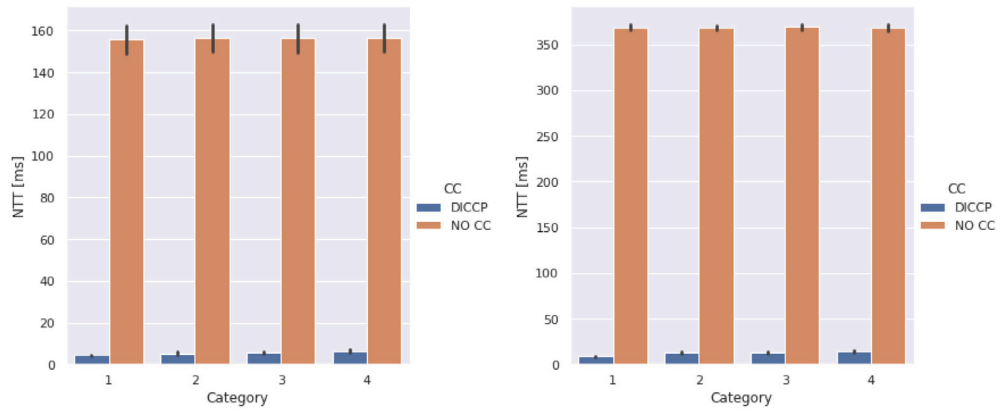
At this point it is worth justifying the throughput values obtained, taking into account that the 802.11ac technology contemplates bit rates that could reach even Gbps values. However, these values depend on several factors, such as the modulation and coding scheme used (called VHT-MCS in 802.11ac), the channel bandwidth (20, 40, 80, 160 MHz) and the number of spatial streams (1, 2, 3). In our simulations, we have chosen (see Table 1) to work with simpler devices, which work correctly with the lowest modulation and coding scheme (VHT-MCS 0), 20 MHz channels, and a single spatial stream. This makes the achievable bit rate around 6.5 Mbps. On the other hand, these 6.5 Mbps represent the rate at which bits can be transmitted over the channel, not the throughput (valid data) that can finally be transmitted or received. The medium access control (MAC) mechanism imposes certain restrictions, such as idle channel time between frames, listening periods prior to the transmission (CSMA/CA algorithm), etc. In addition, transmissions from different nodes may collide, adding back-off times and retransmissions, which also contribute to the reduction of the throughput finally obtained. All these issues are correctly taken into account by the network simulator. On the other hand, the throughput we are presenting is not a one-hop throughput, but the end-to-end throughput in the network. That is, data packets must be retransmitted



(a) Medium Load

(b) High Load

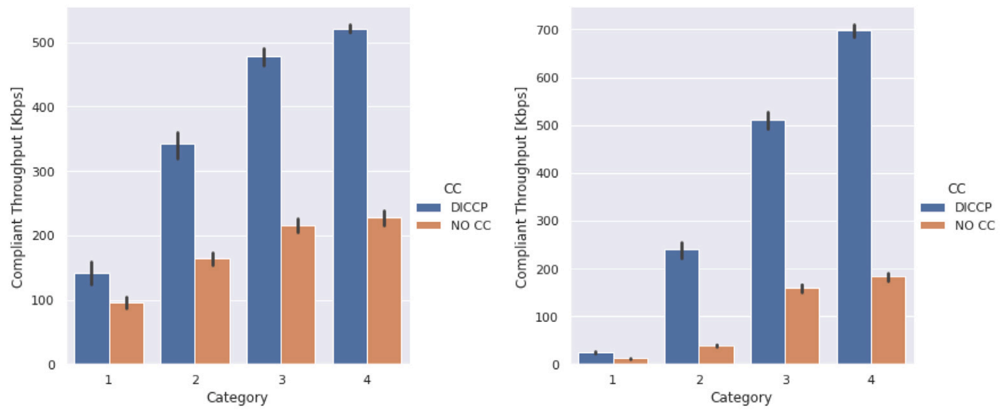
Fig. 12. Packet delivery ratio [%].



(a) Medium Load

(b) High Load

Fig. 13. Network transit time [ms].



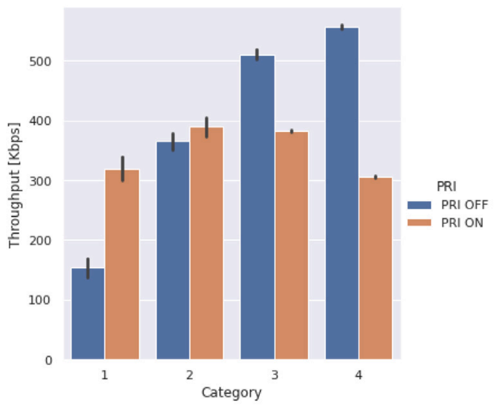
(a) Medium Load

(b) High Load

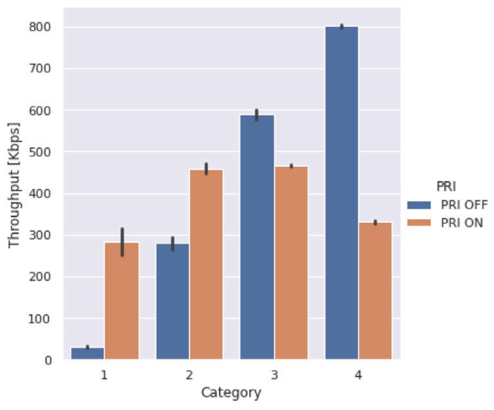
Fig. 14. Compliant received throughput [Kbps].

several times in the multi-hop wireless network until they reach their destination, which also reduces the throughput. Taking all these issues into account, it is reasonable that the total throughput obtained is around 1.5 Mbps, as shown in Fig. 14. Please note that we must add up the throughputs obtained for all categories, since they are all being transmitted over the same medium.

Another relevant issue to take into account is the energy consumption by network nodes, which will decrease with the application of the proposed mechanism. By avoiding the transmission of packets that are not going to be useful at their destination, the consumption generated by the transmission and reception of these packets is saved. In addition, since it is a multi-hop network, these packets could also have been retransmitted by some intermediate nodes, increasing the unnecessary use of energy. On the other hand, it is true that running a new algorithm includes an extra CPU consumption to execute it. However, as we are working with decision trees, the algorithm introduced consists only of

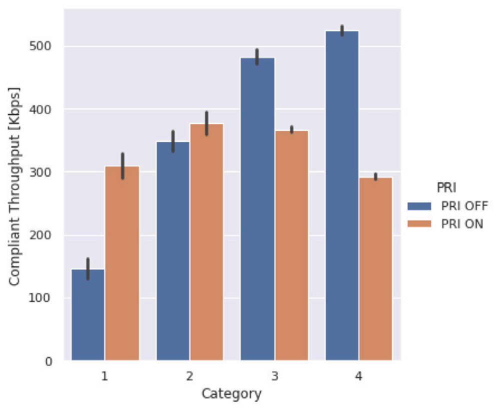


(a) Medium Load

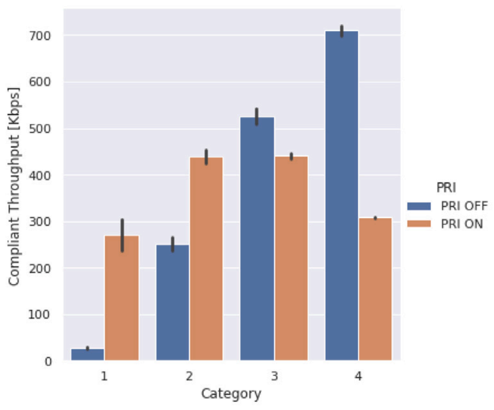


(b) High Load

Fig. 15. Sent throughput [Kbps].



(a) Medium Load



(b) High Load

Fig. 16. Compliant received throughput [Kbps].

a few "if-then-else" steps that do not involve high energy consumption, being negligible compared to the savings obtained by reducing the transmission of packets.

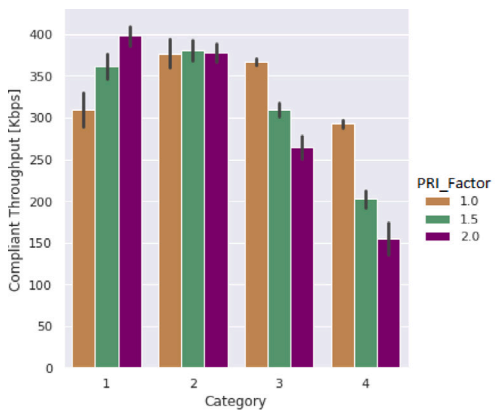
6.3. Priority differentiation

As seen in the previous section, a consequence of the application of the congestion control mechanism is that traffic categories with higher time requirements suffer a greater reduction in their transmission capacity than the categories with less strict requirements. This is because the categories with lower requirements can be transmitted under higher load conditions, and therefore their packets cause the CBR and QP values to remain high. In this way, the amount of time with low CBR and QP values in the network, which is precisely the necessary condition for the successful transmission of packets with high QoS requirements, is reduced.

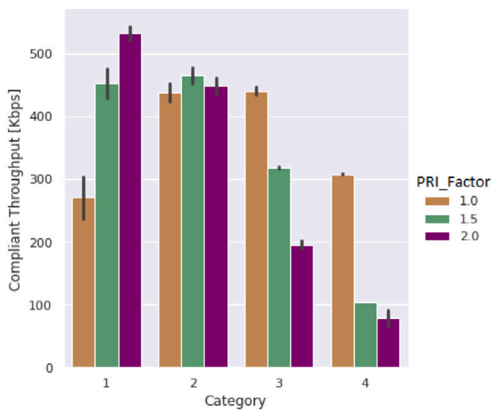
However, the applications with the strongest requirements are often the most critical or relevant in the network, and therefore their packet flows are the most important, i.e., the ones that should be discarded the least. To this end, the prioritization algorithm proposed in Section 5.4 is applied. Fig. 15 presents the results obtained in terms of sent throughput. The chosen value for the priority intensity factor (Eq. (3)) is 1 (default value). The throughput obtained when the congestion control mechanism includes the prioritization algorithm ("PRI ON") is compared with the obtained previously when the prioritization algorithm was not included ("PRI OFF"). For the sake of clarity, the case where no congestion control is applied is not shown anymore. As can be seen, when the priority algorithm is applied, the throughput sent by the categories with the highest requirements (1 and 2) is increased, while the remaining categories (3 and 4) see their throughput reduced. As in previous experiments, this effect is more relevant when the network load is higher. Fig. 16 shows how the compliant received throughput follows the same trend.

At this point, it is interesting to analyze how the priority mechanism allows us to adjust the fairness with which network resources are allocated to different traffic categories. This fairness can be quantified by the Jain's index, defined as $J = 1/(1 + CV^2)$, where CV is the coefficient of variation of the considered variable (ratio between the standard deviation and mean value). In our case the variable will be the compliant throughput of each traffic category. As can easily be seen, the value of the Jain's index is always between 0 and 1, reaching the maximum value when the compliant throughput is the same for all traffic categories (coefficient of variation equal to zero). Without applying the priority mechanism, the value obtained for the Jain's index is 0.863 (medium load) and 0.674 (high load). By applying priorities, the value of the index increases to 0.988 and 0.956 respectively, observing a greater uniformity and fairness in the allocation of resources.

On the other hand, depending on how critical the applications with the highest requirements are for the users, the possibility of increasing the priority assigned to these applications is included. For this purpose, higher values of the priority intensity factor can be selected. The results obtained can be seen in Fig. 17, where the compliant received throughput obtained is shown for three values of this factor (1, 1.5, and 2). As can be seen, the algorithm operates as expected, increasing the compliant throughput for the categories with higher requirements.



(a) Medium Load



(b) High Load

Fig. 17. Compliant received throughput [Kbps].

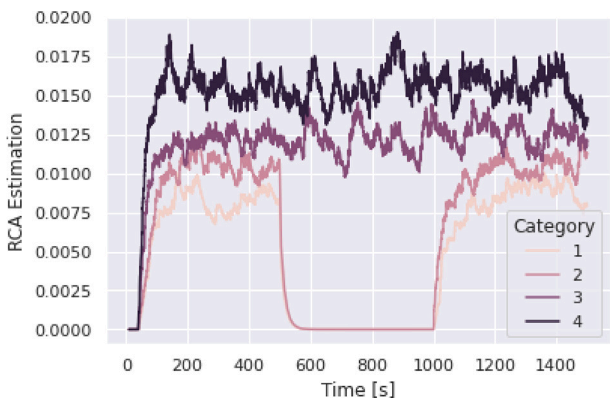


Fig. 18. RCA estimation.

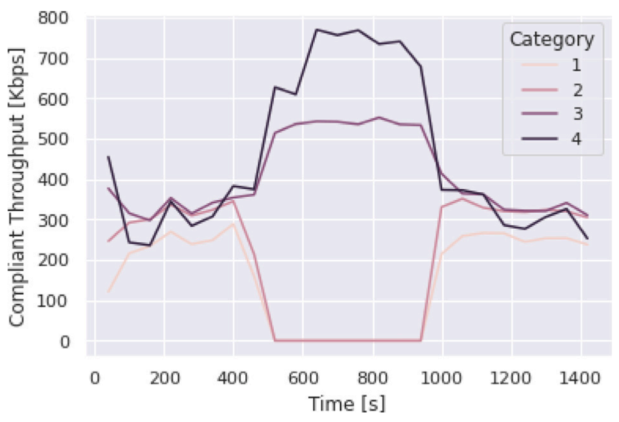


Fig. 19. Compliant received throughput.

6.4. *Dynamic behavior*

Finally, it is relevant to note that all the proposed algorithms adapt dynamically to the network load state. To show this behavior, the results of a simulation in which the load state is variable are presented. Specifically, it is a simulation with a duration of 1500 s. To be more general, the load offered by each of the traffic categories is different (the less restrictive the category, the higher the load). Besides, it is assumed that categories 1 and 2 do not need to transmit during the time interval between 500 and 1000 s. This network load behavior can be seen in Fig. 18, where it is shown the temporal evolution of the estimation made by one of the network nodes (randomly chosen, the estimations made by all the nodes are very similar) of the RCA value of each category.

Fig. 19 shows the temporal evolution of the compliant throughput for each category. To make this figure, the throughput has been calculated in 60-second intervals. As can be seen, due to the absence of traffic belonging to categories 1 and 2 during the time interval [500,1000], the DICCP protocol allocates more transmission resources during this period to categories 3 and 4. When categories 1 and 2 resume their transmissions, the resource allocation returns to the previous situation.

7. Conclusions

In this paper, a new mechanism for congestion control in multi-hop wireless networks has been proposed. The proposal is made for networks as generic as possible, in which all nodes in the network can act as source or destination of data flows (as well as intermediate relay nodes). In addition, the applications that generate the data flows have been differentiated and classified according to two criteria. On the one hand, it is taken into account that the required quality of service (in terms of network transit time) is different for each application. On the other hand, it has been considered that the relevance of each application may also be different.

The proposed solution is based on the application of machine learning techniques, specifically the use of decision trees. The input characteristics of these decision trees are parameters that define the network load state: the channel busy ratio (CBR), and the number of queued packets (QP) waiting in the buffers of the nodes. The output of the trees is the probability that one data packet will reach its destination on time. The trees have been trained from datasets generated by simulations and processed appropriately. Each time an application generates a new data packet to transmit over the network, the node applies the decision tree to predict whether the packet will reach its destination on time. If the prediction is positive, the packet is transmitted; otherwise, it is discarded. In order to apply the decision trees, the nodes need to know the features involved. Thus, these values have to be disseminated in the network, for which the necessary protocol has been designed and implemented.

Various feature configurations have been considered, and it has been found that the best performance is obtained when using both the CBR and the QP of the source, second hop, and destination nodes. With this configuration, different simulations of the complete system have been carried out, verifying the correct operation of the proposal: the data traffics are adequately differentiated according to its time requirements, and relevant improvements are obtained in terms of compliant throughput, i.e., data packets that reach their destination in less time than the maximum limit imposed by their category.

On the other hand, the congestion control mechanism allows different priorities to be assigned to applications, depending on their relevance to the network or to the final users. To this end, based on the transmissions made in each of the categories, an estimation is made of the channel resources required by each application. With this estimation, it is possible to make a resource reservation, starting with the most important category and continuing with the following ones. The simulations carried out corroborate the correct operation of the proposed technique. Finally, it has been proven that the proposed method adapts dynamically to changes in the network load state, caused by variations in the transmission rate of one or more traffic categories.

As a future line of work, other machine learning algorithms will be considered as possible options or extensions of the proposed protocol. For example, reinforcement learning algorithms can be a good option to avoid the need for the dataset generation and model training phase. However, it will be necessary to add some type of feedback between the end nodes of the communication, which can lead to an overload due to this additional control traffic. Finding a good trade-off between the added traffic and the dynamic adaptability of the new model will be the most important part of this job.

Glossary

AUC	Area Under the Curve
CBR	Channel Busy Ratio
DICCP	Distributed Congestion Control Protocol
DICCP_ADP_PRI	DICCP Application Data Packet Priority
DICCP_CAT	DICCP Categories
DICCP_EWMA_W	DICCP Exponential EWMA Weight
DICCP_MIT	DICCP Message Interval Time
DICCP_NThres	DICCP Neighbors Threshold
DICCP_NVT	DICCP Neighbors Valid Time
DICCP_PRI	DICCP Priority
DICCP_SIT	DICCP Sample Interval Time
DCN	Data Center Network
ECN	Explicit Congestion Notification
EDCA	Enhanced Distributed Channel Access
EWMA	Exponentially Weighted Moving Average
FPR	False Positive Rate
GRCA	Global Required Channel Availability
IP	Internet Protocol
ML	Machine Learning
NBH	Neighborhood Table
NIC	Network Interface Card
NN	Number of Neighbors
NST	Network State Table
NTT	Network Transit Time
QoS	Quality of Service
QP	Queued Packets
RCA	Required Channel Availability
RCAT	RCA Estimation Table
ROC	Receiver Operating Characteristic
RTT	Round Trip Time
TCP	Transmission Control Protocol
TPR	True Positive Rate
UDP	User Datagram Protocol
VHT-MCS	Very High Throughput Modulation and Coding Scheme
WAN	Wide Area Network

CRediT authorship contribution statement

Juan Pablo Astudillo León: Conceptualization, Investigation, Formal analysis, Methodology, Software, Data curation, Visualization, Validation, Writing – original draft, Writing – review & editing. **Luis J. de la Cruz Llopis:** Conceptualization, Investigation, Formal analysis, Methodology, Software, Data curation, Visualization, Validation, Supervision, Funding acquisition, Resources, Writing – original draft, Writing – review & editing, Project administration. **Francisco J. Rico-Novella:** Conceptualization, Investigation, Formal analysis, Methodology, Software, Data curation, Visualization, Validation, Supervision, Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

The Spanish Government supported this work under the research project "Enhancing Communication Protocols with Machine Learning while Protecting Sensitive Data (COMPROMISE)" (PID2020-113795RB-C31/AEI / 10.13039/501100011033).

References

[1] D. Reina, M. Askalani1, S. Toral, F. Barrero, E. Asimakopoulou, N. Bessis, A survey on multihop ad hoc networks for disaster response scenarios, Int. J. Distrib. Sens. Netw. 11 (10) (2015).

[2] A. Srivastava, A. Prakash, R. Tripathi, A cross layer based cooperative broadcast protocol for multichannel VANET, Ad Hoc Netw. 131 (2022) 102840, http://dx.doi.org/10.1016/j.adhoc.2022.102840.

[3] H. Nawaz, A.H. Mansoor, A.A. Laghari, UAV communication networks issues: A review, Arch. Comput. Methods Eng. 28 (2021) 1349–1369, http://dx.doi.org/10.1007/s11831-020-09418-0.

[4] Y. Azzoug, A. Boukra, Enhanced UAV-aided vehicular delay tolerant network (VDTN) routing for urban environment using a bio-inspired approach, Ad Hoc Netw. 133 (2022) 102902, http://dx.doi.org/10.1016/j.adhoc.2022.102902.

[5] K.-Y. Tsao, T. Girdler, V.G. Vassilakis, A survey of cyber security threats and solutions for UAV communications and flying ad-hoc networks, Ad Hoc Netw. 133 (2022) 102894, http://dx.doi.org/10.1016/j.adhoc.2022.102894.

[6] I. Ahmad, S. Shahabuddin, H. Malik, E. Harjula, T. Leppänen, L. Loven, A. Anttonen, A.H. Sodhro, M.M. Alam, M. Juntti, et al., Machine learning meets communication networks: Current trends and future challenges, IEEE Access 8 (2020) 223418–223460.

[7] L. Lemus Cárdenas, A.M. Mezher, J.P. Astudillo León, M. Aguilar Igartua, DTMR: A decision tree-based multimetric routing protocol for vehicular ad hoc networks, in: Proceedings of the 18th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, 2021, pp. 57–64.

[8] L. Lemus Cárdenas, A.M. Mezher, P.A. Barbecho Bautista, J.P. Astudillo León, M.A. Igartua, A multimetric predictive ANN-based routing protocol for vehicular ad hoc networks, IEEE Access 9 (2021) 86037–86053.

[9] L.L. Cárdenas, J.P. Astudillo León, A.M. Mezher, GraTree: A gradient boosting decision tree based multimetric routing protocol for vehicular ad hoc networks, Ad Hoc Netw. 137 (2022) 102995.

[10] C.L. Duenas Santos, J.P. Astudillo León, A.M. Mezher, J. Cardenas Barrera, J. Meng, E. Castillo Guerra, RPL+: An improved parent selection strategy for RPL in wireless smart grid networks, in: Proceedings of the 19th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, 2022, pp. 75–82.

[11] T. Clausen, P. Jacquet, Optimized link state routing protocol (OLSR), Internet Eng. Task Force (IETF) 4 (2003) 75, doi:10.1.1.11.620, arXiv:arXiv:1011.1669v3.

[12] T. Clausen, C. Dearlove, P. Jacquet, U. Herberg, The Optimized Link State Routing Protocol Version 2, Tech. Rep. RFC 7181, Internet Engineering Task Force (IETF), 2014, https://www.rfc-editor.org/info/rfc7181.

[13] B.A.T.M.A.N., 2022, https://www.open-mesh.org/projects/open-mesh/wiki.

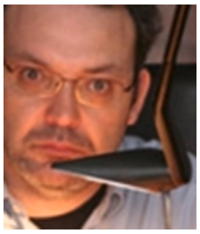
[14] C. Tripp-Barba, L. Urquiza-Aguiar, M.A. Igartua, D. Rebollo-Monedero, L.J. de la Cruz llopis, A.M. Mezher, J.A. Aguilar-Calderón, A multimetric, map-aware routing protocol for VANETs in urban areas, Sensors 14 (2014) 2199–2224, http://dx.doi.org/10.3390/s140202199.

[15] M. Joseph Auxilius Jude, V.C. Diniesh, M. Shivaranjani, Throughput stability and flow fairness enhancement of TCP traffic in multi-hop wireless networks, Wirel. Netw. 26 (2020) 4689–4704, http://dx.doi.org/10.1007/s11276-020-02357-5.

[16] J.P. Astudillo León, L.J. de la Cruz Llopis, Emergency aware congestion control for smart grid neighborhood area networks, Ad Hoc Netw. 93 (2019).
[17] J.P. Astudillo León, T. Begin, A. Busson, L.J. de La Cruz Llopis, A fair and distributed congestion control mechanism for smart grid neighborhood area networks, Ad Hoc Netw. (2020) http://dx.doi.org/10.1016/j.adhoc.2020.102169, URL http://www.sciencedirect.com/science/article/pii/S1570870520300974.
[18] G. Carlucci, L. De Cicco, S. Holmer, S. Mascolo, Analysis and design of the google congestion control for web real-time communication, in: Proceedings of the 7th International Conference on Multimedia Systems, WebRTC, 2016, pp. 1–12.
[19] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), in: Proceedings of the ACM SIGCOMM 2010 Conference, 2010, pp. 63–74.
[20] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, G. Judd, Data Center TCP (DCTCP): TCP Congestion Control for Data Centers, Tech. Rep., 2017.
[21] M. Zhang, Y. Zhu, J. Zhang, Y. Mao, Y. Liu, DCQCN: Enabling congestion control for datacenter networks, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012, pp. 157–170.
[22] Y. Zhang, A. Kabbani, M. Ghobadi, M. Chiang, P. Mishra, J. Gottlieb, DCQCN: Achieving near-optimal queue length and low delay for data center applications, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets-XV, ACM, 2016.
[23] P. Mittal, V. Vasudevan, P. Sharma, Swift: Fast, cheap, and fair congestion control for the datacenter, in: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, 2015, pp. 267–280.
[24] B. Li, H. Cheng, Y. Wang, H. Zhang, SWIFT: A fast TCP variant for data center networks, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016.
[25] B. Montazeri, Y. Li, M. Alizadeh, J. Ousterhout, Homa: A receiver-driven low-latency transport protocol using network priorities, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 221–235.
[26] IETF, QUIC: A UDP-Based Multiplexed and Secure Transport, Tech. Rep., RFC 9000, 2020, URL https://tools.ietf.org/html/rfc9000.
[27] A. Brunstrom, T. Edmonds, C.B. Margaria, An evaluation of TCP and QUIC for web traffic, in: 2020 IFIP Networking Conference, Networking, IEEE, 2020, pp. 144–152.
[28] J. Iyengar, I. Swett, QUIC Loss Detection and Congestion Control, Tech. Rep., IETF RFC 9002, 2021, URL https://tools.ietf.org/html/rfc9002.
[29] T. Henderson, S. Floyd, A. Gurtov, Y. Nishida, The NewReno Modification to TCP's Fast Recovery Algorithm, Tech. Rep., IETF RFC 6582, 2012, URL https://tools.ietf.org/html/rfc6582.
[30] Y. Sun, M. Peng, Y. Zhou, Y. Huang, S. Mao, Application of machine learning in wireless networks: Key techniques and open issues, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3072–3108.
[31] H. Jiang, Q. Li, Y. Jiang, G. Shen, R. Sinnott, C. Tian, M. Xu, When machine learning meets congestion control: A survey and comparison, Comput. Netw. 192 (2021) 108033, http://dx.doi.org/10.1016/j.comnet.2021.108033, URL https://www.sciencedirect.com/science/article/pii/S1389128621001407.
[32] S. Abbasloo, C.-Y. Yen, H.J. Chao, Classic meets modern: A pragmatic learning-based congestion control for the internet, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '20, 2020, pp. 632–647, http://dx.doi.org/10.1145/3387514.3405892.
[33] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, Y. Zhu, L. Cui, Congestion control for cross-datacenter networks, in: 2019 IEEE 27th International Conference on Network Protocols, ICNP, 2019, pp. 1–12, http://dx.doi.org/10.1109/ICNP.2019.8888042.
[34] J.P. Astudillo León, F.J. Rico-Novella, L.J. de la Cruz Llopis, Predictive traffic control and differentiation on smart grid neighborhood area networks, IEEE Access 8 (2020) 216805–216821, http://dx.doi.org/10.1109/ACCESS.2020.3041690.
[35] ns-3 network simulator, 2022, https://www.nsnam.org/.
[36] E. Khorov, A. Kiryanov, A. Lyakhov, G. Bianchi, A tutorial on IEEE 802.11ax high efficiency WLANs, IEEE Commun. Surv. Tutor. 21 (1) (2019) 197–216, http://dx.doi.org/10.1109/COMST.2018.2871099.
[37] V.L. Parsons, Stratified sampling, in: Wiley StatsRef: Statistics Reference Online, Wiley Online Library, 2014, pp. 1–11.
[38] P.L. Barreiro, J.P. Albandoz, Population and sample. Sampling techniques, Manag. Math. Eur. Schools 1 (1) (2001) 1–18.
[39] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning: Methods, Systems, Challenges, Springer Nature, 2019.
[40] Y. Sasaki, et al., The truth of the F-measure, Teach Tutor. Mater. 1 (5) (2007) 1–5.
[41] G. Hackeling, Mastering Machine Learning with Scikit-Learn, Packt Publishing Ltd, 2017.
[42] M.L. McHugh, Interrater reliability: The kappa statistic, Biochem. Med. 22 (3) (2012) 276–282.
[43] S. Raschka, Python Machine Learning, Packt Publishing Ltd, 2015.
[44] L. Prokhorenkova, G. Gusev, A. Vorobev, A.V. Dorogush, A. Gulin, CatBoost: Unbiased boosting with categorical features, Adv. Neural Inf. Process. Syst. 31 (2018).
[45] Y. Yao, L. Rosasco, A. Caponnetto, On early stopping in gradient descent learning, Constr. Approx. 26 (2) (2007) 289–315.
[46] L. Prechelt, Early stopping — But when? in: G. Montavon, G.B. Orr, K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade, second ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 53–67.
[47] G. Raskutti, M.J. Wainwright, B. Yu, Early stopping and non-parametric regression: An optimal data-dependent stopping rule, J. Mach. Learn. Res. 15 (1) (2014) 335–366.
[48] J. Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Measur. 20 (1) (1960) 37–46.

Juan Pablo Astudillo León received the Ph.D. in network engineering (Hons.) from the Universitat Politècnica de Catalunya (UPC), Spain, in 2020. He received the Best Ph.D. Thesis Prize on information and communication technologies from the Escola de Doctorat (UPC). He was a post-doctoral fellow at the University of New Brunswick (UNB), Canada. He is currently a full-time Professor with Yachay Tech University, Urcuquí, Ecuador and Postgraduate Professor at the Pontificia Universidad Católica del Ecuador (PUCE), Manabí. He worked as a Undergraduate Professor at the Universidad Politécnica Salesiana (UPS), Ecuador. He has been involved in several national and international projects and has authored international publications in conferences and journals. His research explores the application of artificial intelligence in wireless multi-hop networks and also the development of IoT smart services.

Luis J. de la Cruz Llopis received the telecommunication engineering degree in 1994 and the Ph.D. in telecommunications engineering in 1999, both from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is currently an Associate Professor at the Department of Network Engineering of the UPC. He is part of the SISCOM (Smart Services for Information Systems and Communication Networks) research group, within which he participates in research projects in collaboration with various public universities, and collaborates with private companies in the development of technological communications projects. His current research interests include the application of machine learning techniques in wireless multi-hop and C-V2X networks, and also the development of IoT smart services.



Francisco J. Rico-Novella received the telecommunication engineering degree in 1989 and the Ph.D. in telecommunications engineering in 1995, both from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is currently an Associate Professor at the Department of Network Engineering of the UPC. His current research interests include cryptography and IoT smart services.