Network- and Application-aware Adaptive Congestion Control Algorithm

Ramyashree Venkatesh Bhat, Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke

***Abstract*—More and more traffic is migrating to private networks in use in various professional environments. With this, there comes a growing diversity in applications, each of them requiring different quality of service. This poses challenges to properly managing such networks, which can span both wired and wireless segments. To tackle the issue of network management and application demands in such networks, we introduce a network- and application-aware adaptive congestion control algorithm, which provides congestion-free service differentiation to the flows in wireless networks in a decentralized manner. The designed algorithm operates based on the real-time network information obtained from in-band network telemetry and aggregated flow information from intermediate nodes. The algorithm performs three times better than the existing CUBIC congestion control algorithm and twice better in a multi-flow architecture. The designed algorithm is the first step towards adaptive transport and application layer protocols which are the future of private professional networks.**

***Index Terms*—APP-NET, congestion control algorithm, connectivity, eBPF, INT, wireless networks.**

I. Introduction

OVER the years, there has been a tremendous increase in the number of connected devices, more and more of them becoming part of private networks found for instance in industrial settings, office environments or other professional environments. Such networks are also more open to innovations compared to innovations targeting the scale of the Internet. The applications running in these networks can impose quite diverse and stringent requirements that must be satisfied by the underlying communication infrastructure, resulting in different traffic types demanding different quality of services (QoS). When we look at the current state of the network, applications are confined to choosing QoS offered by differentiated services (DiffServ) and when the number of flows in the network increases, say in the range of 100s and 1000s, achieving fine-grained DiffServ for each flow becomes harder. To overcome these issues, deterministic[1] and time-sensitive[2] task groups are aiming to achieve congestion-free operation through slicing and scheduling by exploiting the flexibility of programmable data planes. However, such systems can only accommodate a limited number of slices and schedules, leading to little flexibility and more complications in networks where multiple flows of different priorities coexist. As a result, when several flows with lower priority get assigned to the same slice, it might eventually result in congestion in the network or into a common treatment of flows with diverse needs. Therefore, there is a need to complement such mechanisms with decentralized approaches, able to adapt to the growing network traffic, at the same time taking into account the priorities of emerging applications and achieving congestion-free differentiation of services. The transmission control protocol (TCP), is an example of such a decentralized approach offering connection-oriented, reliable, and error-free data transfer. However, as it only has an end-to-end view of the network, it is only partially able to achieve the aforementioned goals.

A recent development in networking, application-network (APP-NET) integration, has enabled applications to specify their traffic and monitoring requirements to the network layer [3]. This provides a possibility to involve applications in impacting network management by acknowledging its requirements to the network layer. Another recent innovation is continuous in-band monitoring and verification technique called in-band network telemetry (INT). INT is capable of collecting node characteristics along with wireless link information in real-time which can be accessed by the application layer [4]. Previously in [5], we have used INT and APP-NET to modify the existing CUBIC congestion control algorithm for wireless networks. The obtained results indicate that utilizing INT information can greatly improve the overall performance of the congestion control algorithm. Therefore, in this study, we address the problem of congestion and differentiation of services by designing a rule-based network- and application-aware adaptive congestion control algorithm (NACC). This will be a stepping stone in designing adaptive transport and application layer protocols, that more optimally make use of the available network resources and are in line with their needs.

The key research contributions of this study are as follows:

1) Design of a rule-based NACC, that operates based on real-time network context and aggregated flow information to achieve congestion free differentiation in services

Manuscript received May 23, 2023; revised September 8, 2023; approved for publication by Lee, SuKyoung, Division 3 Editor, October 31, 2023.

This research was partially funded by the FWO-Flanders under grant agreements #S003921N and #G055619N and by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie Vlaanderen" program. This work was also partially supported by the CHIST-ERA grant SAMBAS (CHIST-ERA-20-SICT-003), with funding from FWO, ANR, NKFIH, and UKRI.

The authors are with the Faculty of Engineering and Architecture Ringgold standard institution, Ghent University, Gent, Belgium, email: {ramyashreevenkatesh.bhat, jetmir.haxhibeqiri, ingrid.moerman, jeroen.hoebeke}@ugent.be.

Digital Object Identifier: 10.23919/JCN.2023.000052

[1]https://datatracker.ietf.org/wg/detnet/about/

[2]https://1.ieee802.org/tsn/

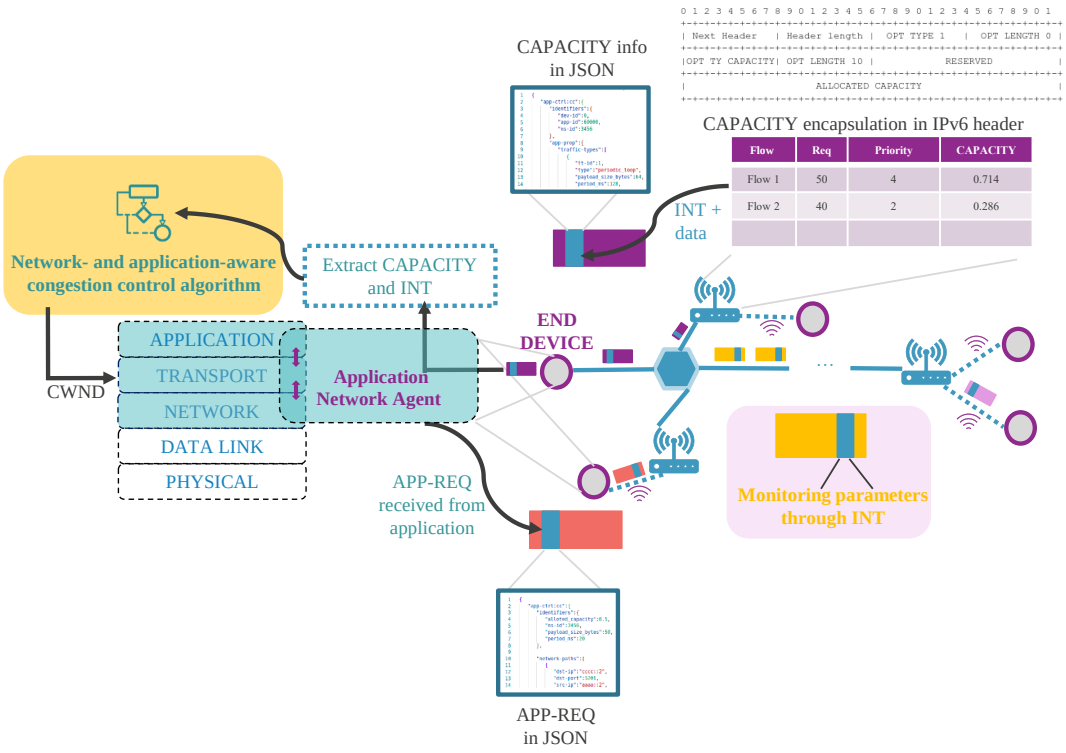


Fig. 1. Network architecture overview.

in private professional networks that support in-band telemetry and feedback.

2) Performance evaluation of the designed algorithm in comparison with the existing CUBIC congestion control algorithm in terms of stability, throughput, fairness and application priorities.

The rest of the paper is structured as follows, Section II discusses the related works in the area of transport protocols. Section III discusses the goals and Section IV provides the network architecture overview. Section V describes the design and implementation of network-aware congestion control algorithm. The design and implementation of NACC is elaborated in Section VI. Section VII presents the results and discussions of the performance evaluation of NACC. Section VIII concludes the article with insights on future works.

II. Related Work

Over the years there has been several works to improve mechanisms of transport layer protocol. The Accurate-ECN [10] is designed as an enhancement to explicit congestion notification (ECN) by utilizing INT for wired networks. The authors do not yet utilize the monitoring information to decide the data transfer rate nor take into account the challenges of wireless networks. The authors of [11] have designed a multi objective congestion control algorithm that adapts itself to different application requirements and corresponding optimal control policies. The congestion control algorithm designed in this work is a reinforcement learning based algorithm which is capable of transferring its knowledge from past experience to new applications and doesn't take into account the real-time network state information. A queuing delay variation-based adaptive congestion control algorithm has been designed by the authors of [12]. The algorithm obtains an optimal congestion window size based on cross-layer-based initialization method, adapts to the network conditions to reduce packet losses and retransmissions and tries to achieve equal fairness among different flows. Other systems such as BANQUET [8] adjust the bitrate by predicting the QoE and traffic volume based on future throughput and a buffer transition calculation but is mainly intended for multimedia applications. Google's QUIC aims to provide low-latency data transfer to online applications [2]. The spin bit feature of QUIC enables latency monitoring, which is rather reactive than proactive with a partial network view. Sextant is a system that has been designed to enable network-aware application optimization in carrier networks [6]. This approach does not yet consider the congestion and detailed QoS specific to each application. Also, authors in [7] have designed adaptive packet transmission techniques as a response to an abnormality in the software-defined smart meters. The PANAPI system is being designed to provide an API for applications to choose from the available network paths in a path-aware environment [9]. The main disadvantage of this system is the requirement for a network back-end database with information about assessed path qualities. The literature mentioned in this section either utilizes partial network information in the congestion control algorithms or operates in a centralized network architecture, hence requiring more computational power. These works do not yet take into account the issue of providing differentiated

services to the applications based on their requirements nor address the issue of congestion control in wireless networks at the same time. Only one or two of these issues are considered in each work. The designed algorithm NACC addresses the issue of congestion-free service and service differentiation in wired-wireless network architecture. NACC utilizes real-time network information to provide congestion-free data transfer and achieves the fulfillment of QoS requirements based on application requirements in a decentralized network architecture.

III. Goals

To achieve **congestion-free differentiation of services**, we have considered certain goals in designing NACC. The NACC should not wait until the packet loss to take an action, it has to be **proactive**. The algorithm should quickly reach the maximum possible data transfer rate and fully utilize the available bandwidth. The algorithm should maintain a **stable data transfer rate**, achieving better throughput. The algorithm should be able to **differentiate the packet losses** due to congestion and poor quality of the network. The algorithm should **differentiate the data transfer rate** among different flows based on the application priorities unless achieving fairness in case of the same priorities.

IV. Network Architecture and Enablers

To realize the above mentioned goals, we need the network architecture that provides continuous monitoring data, application network interaction and a framework to reconfigure the congestion control algorithm. These three frameworks are indicated in the Fig. 1 as (a), (b), and (c) and explained in this section.

A. Monitoring Parameters Through INT

To achieve stable data transfer rate and differentiate the packet losses it is necessary to continuously receive the real time network data. Hence we use INT which has been extended to wireless networks by designing new INT-enabled node architecture and logic to process INT-enabled packets for WiFi-based networks [4]. This design is capable of collecting node characteristics such as queue information, processing delay, and Tx/Rx timestamping values, along with wireless link information (such as data rate, received signal strength (RSSI), signal to noise ratio (SNR), the channel used, etc.) and end-to-end flow characteristics (flow latency, flow jitter and flow packet loss ratio) (Fig. 1). The INT information is encapsulated in IPv6 extension header[3][4] and is received as a feedback at the source node. Hence it is a low overhead, continuous network monitoring technique which provides real-time network context to reconfigure the data transfer rate.

[3]https://p4.org/p4-spec/docs/INT_v2_1.pdf
[4]https://tools.ietf.org/html/draft-ietf-ippm-ioam-data-04#section-4

B. Framework for Application and Network Interaction

To achieve our goals the NACC needs to understand the application requirements as well as real time network data obtained from INT. Hence we use the APP-NET framework designed in [3], in which the application's data plane and control plane are integrated through an application network agent (ANA) which lies between the application and the network stack (Fig. 1). This design has a network and system-independent APP-ANA interface and a network stack-specific ANA-NET interface. This framework is capable of passing application requirements (APP-REQ) such as application identifiers (device ID, application ID, and node ID) and application properties (traffic types, payload size, periodicity, priority (PRIORITY), and network paths), at the same time passing monitored performance of the traffic flow as well as monitoring feedback from the other end node from the network layer to the application. The framework also provides the possibility to further extend the interface to the transport layer through transport adapter resulting in application-transport-network layer interaction. The APP-REQ is modeled as a JSON data structure and is encapsulated in the IPv6 extension header. This information is sent as INT from the source to the destination during the connection initiation i.e., in the SYN packet.

C. Reconfigurability of Congestion Control Algorithms

In order to design NACC, we use congestion control plane (CCP) API [13] which provides a framework to reconfigure the congestion control algorithms. It allows us to write congestion control algorithms in the Python programming language and run them in any environment.

V. Network-awareness: Design and Implementation

A. Relevant Monitoring Parameters

Before designing the congestion control algorithm based on INT, it is important to revisit the relationship between different parameters obtained from INT and the congestion window size (CWND) of TCP [5]. In existing congestion control algorithms, with the increase of the CWND, the buffer queue and the load at the interface of the intermediate device increases as well. With the increase in the number of flows in the network the congestion is increased too. The lower the data rate of the link, the lesser the amount of data that can be sent through the network. The quality of link is also partially indicated by the round trip time of the packets. The existing TCP congestion control algorithms cannot differentiate between the packet loss due to the actual network congestion and poor wireless connection, hence every packet loss due to the poor quality of the network also affects the CWND. Therefore, of the different parameters collected through INT, available queue capacity (Q), load at the interface (data arrival rate, A), flow count (FC), wireless packet loss (W_PL), and varying wireless link data rate (DR) are the ones that directly affect the congestion in a network and thus CWND is related to these parameters. The designed congestion control algorithm

TABLE I
SUMMARY OF ABBREVIATIONS, THEIR MEANINGS, AND UNITS.

Abbreviation	Meaning	Unit	Source
CWND	Congestion window size	packets	calculated
Q	Available queue capacity	packets/second	INT
A	Data arrival rate	bits/second	INT
FC	No. of flows passing through the network node	no. of flows	INT
W_PL	Packet loss due to the quality of wireless medium	packets	INT
DR	Data rate of the link	Mbps	INT
RTT	Round trip time	second	kernel
MSS	Maximum segment size	bytes	kernel
CWND_L	CWND does not go below this lower bound	packets	calculated
CWND_UP	CWND does not exceed this upper bound	packets	packets
A_UP	Data arrival rate does not exceed this upper bound	bits/second	calculated
B_CWND	CWND of previous data transfer	packets	calculated
A_DIFF	Rate of increase in the load at the network node	percentage	calculated
PRIORITY	Required application priority	-	APP_REQ
MAX_PRIORITY	Total no. of available priority levels	-	APP_REQ
CAPACITY	Percentage of DR available for data transfer	percentage	INT

gets real-time updates on these parameters. Suppose there are several intermediate nodes in the network, then the bottleneck value of these parameters is considered.

B. Design of Network-aware Congestion Control Algorithm

While sending the SYN packet, the CWND is set to the same value as *ssthresh* in conventional congestion control algorithms of TCP. After receiving the first SYN-ACK packet, the initialization phase of the designed algorithm will set the CWND's lower (CWND_L) and upper bounds (CWND_UP) and data arrival rate upper bound (A_UP) as per the (1), (2), and (3), respectively. The value of CWND_L is equal to the *ssthresh*. A_UP is the upper bound above which the data arrival rate at the intermediate node does not exceed. The A_UP is set to 16% of DR. This 16% is obtained by considering the percentage of DR that is utilized for data transfer (25%)[5], overhead added by the lower layers (5%) (gives 20% of DR), and a margin of 20% from 0.2·DR (in case a new flow wants to initiate a connection). CWND_UP being the upper bound of CWND is calculated using 16% of DR (16% is the combined overhead by lower layers and 20% margin as mentioned previously), round trip time (RTT), and maximum segment size (MSS) (MSS is multiplied by 8 to convert bytes to bits) of the packets. The upper bounds are updated with changing link DR and RTT obtained from INT in real time.

[5]https://www.itweb.co.za/content/o1Jr5qx96jDvKdWL

The DR with overhead indicates the amount of bandwidth available for data transfer, hence the AR at the intermediate node cannot exceed this value, at the same time, the data sent by the source should be kept below this DR with overhead to avoid congestion.

$$CWND\_L = 10 \tag{1}$$

$$CWND\_UP = CWND\_L + \frac{(DR \cdot 0.16) \cdot RTT}{8 \cdot MSS} \tag{2}$$

$$A\_UP = DR \cdot 0.16 \tag{3}$$

Generally, congestion control algorithms have different phases such as slow start, congestion avoidance, fast recovery, and fast retransmit. As mentioned in Section III, our aim is to fully utilize the available capacity and reach a threshold that will sustain the data transfer rate and achieve better throughput. Our algorithm has two main states, one on receiving an acknowledgment (or selective acknowledgment) notification (ON_ACK) and the second on detecting a packet loss (ON_LOSS). Everytime the algorithm receives one of these notifications, it reads the real-time DR data from INT and sets CWND_UP and A_UP.

ON_ACK, the algorithm first checks the FC feedback (FC_UPDATE) obtained from INT. If the number of flows in the network changes, then the previous

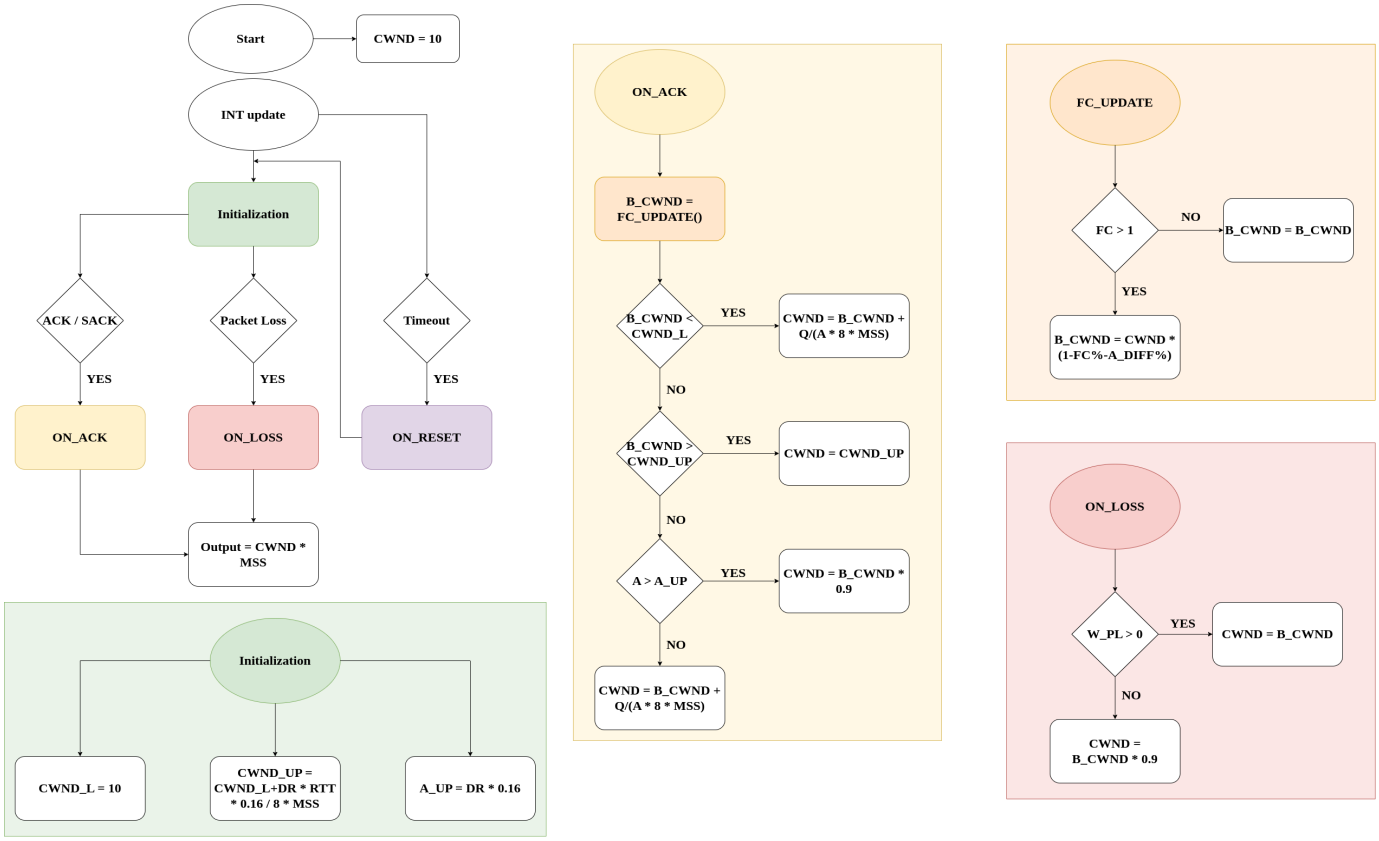


Fig. 2. Flowchart of network-aware congestion control algorithm.

CWND (B_CWND) is reduced by the percentage of FC and A_DIFF (4). A_DIFF is the rate of increase in the load at the intermediate node. Once the B_CWND is set based on FC, the algorithm will compare its value with various thresholds. Starting with CWND_L, if the B_CWND is lower than this threshold, then the CWND is aggressively increased until it exceeds CWND_L (6). Then the B_CWND is compared with CWND_UP and if B_CWND is greater than CWND_UP, then the CWND is the same as the CWND_UP. If not, then the current A is compared with the A_UP. If A is greater than A_UP, it means that the load on intermediate nodes is more than the available DR, hence the CWND is decreased by 10% (justified in the next paragraph) (5). If not, then CWND is increased proportionally to the Q, A, and 1/RTT (6), thus achieveing stable CWND. Once the maximum possible CWND for the specific flow under given network is achieved, the algorithm is designed to maintain the same CWND (or at least vary in a small window size) throughout the data transfer.

$$B\_CWND = CWND \cdot \ (1 - FC\% - A\_DIFF\%) \tag{4}$$

$$CWND = B\_CWND \cdot 0.9 \tag{5}$$

$$CWND = B\_CWND + \frac{Q}{A \cdot 8 \cdot MSS} \tag{6}$$

ON_LOSS, the algorithm checks if the loss is due to the poor quality of the wireless network or due to congestion. If the loss is due to the poor quality of the network, then the CWND is kept constant since it has better performance than reducing the CWND [5]. The conventional congestion control algorithm reduces the CWND by 30% on packet loss irrespective of the reason for the packet loss. We reduced CWND by 30%, 20%, 10%, and 5% on packet loss and tested it in a network with congestion. The overall data throughput was better when the CWND was reduced by 10%, hence both CWND and CWND_UP are reduced by 10% when the packet loss is due to congestion (7).

$$CWND = B\_CWND \cdot 0.9 \tag{7}$$

In case of TCP connection timeout, the algorithm enters the ON_RESET function, which is the same state as when a new connection starts, i.e., by sending the SYN packet. Then, the CWND is set to the value of 10, which is equal to *ssthresh* in conventional congestion algorithms. If the SYN-ACK packet is received, the algorithm enters the initialization phase where the values of CWND_L, CWND_UP, and A_UP are re-initialized as per the (1), (2), and (3), respectively.

The above explained algorithm is indicated as a flow chart/algorithm in Fig. 2.

The designed network-aware congestion control algorithm uses real-time monitoring information obtained from INT [4] The design was implemented in Python 3.7 programming language using CCP [13].

```
1  {
2      "app-ctrl:cc":{
```

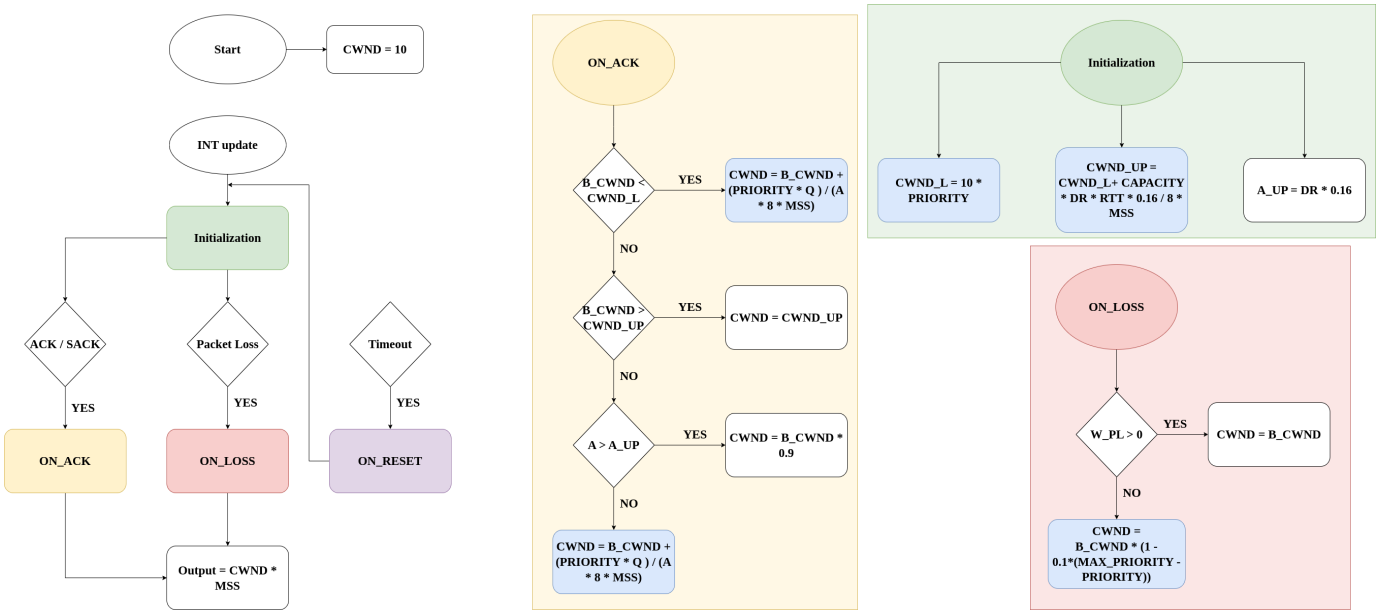


Fig. 3. Flowchart of network- and application-aware congestion control algorithm.

```
        "identifiers":{
            "calculated_capacity":0.5,
            "ns-id":3456,
            "payload_size_bytes":50,
            "period_ms":20
        },

        "network-paths":[
            {
                "dst-ip":"cccc::2",
                "dst-port":5201,
                "src-ip":"aaaa::2",
                "src-port":"52296"
            }
        ]
    }
}
```

VI. Application Awareness in Network-aware Congestion Control Algorithm

Given a network with one flow, the application can fully utilize the available bandwidth for data transfer. When there are several other flows in the network, the conventional congestion control algorithm of TCP tries to achieve fairness between the flows irrespective of their priority and traffic needs. We aim to design an algorithm that provides fairness based on APP-REQ of the flows [Section III]. This section elaborates on how we achieve application awareness in the network-aware congestion control algorithm.

A. APP-REQ Processing in Intermediate Nodes: Data Encapsulation and Extraction

As explained in Section IV-B, APP-REQ is sent as INT from the source to the destination during the connection initiation i.e., in the SYN packet. The intermediate nodes decapsulate this APP-REQ and store the flow information along with its traffic properties. The application identifiers help to differentiate between different flows of different applications from the same node. The intermediate nodes calculate the available capacity (percentage of data rate available for data transfer) to this flow. Whenever the node receives a packet destined to the source node (for a new flow it is SYN-ACK), it embeds the calculated capacity (CAPACITY) information in the units of percentage (in decimal format) (as calculated in (8)) in JSON format along with the application identification as shown in the Listing V-B. Equation 8 calculates the capacity, CAPACITY_r of flow r with $payload_r$, $period_r$ and priority p_r requirements and n flows in the node. This data structure is encapsulated in the IPv6 extension header [Fig. 1] and the source node can extract the CAPACITY information from the INT packet.

$$CAPACITY_r = \frac{p_r \cdot \left(\frac{payload_r}{period_r}\right)}{p_1 \cdot \left(\frac{payload_1}{period_1}\right) + \cdots + p_n \cdot \left(\frac{payload_n}{period_n}\right)} \tag{8}$$

Suppose multiple flows are already passing through an intermediate node, whenever a new flow initiates the connection, the intermediate then distributes the available capacity among different flows (8) and updates the capacity information of other existing flows along with the new flow. The node then waits for the packets that are destined for source nodes (including the packets for nodes with previously existing flows) and notifies all the source nodes whose flows pass through the intermediate node with the updated capacity allocation. By sharing just the capacity allocation information, the intermediate node does not give out the application identification of other flows in the network and thus protects the privacy of the network.

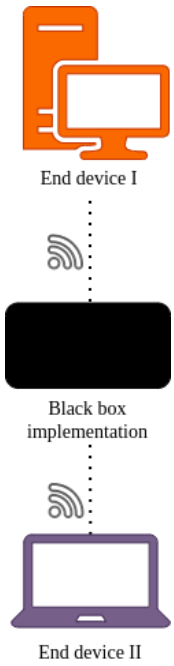


Fig. 4. Network emulated in Mininet-wifi, with black box implementation.

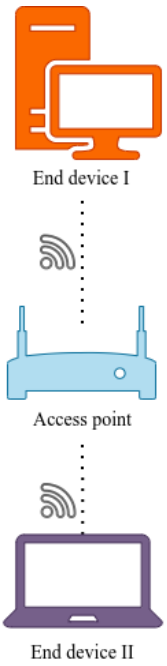


Fig. 5. Network emulated in Mininet-wifi, with an access point and two end devices.

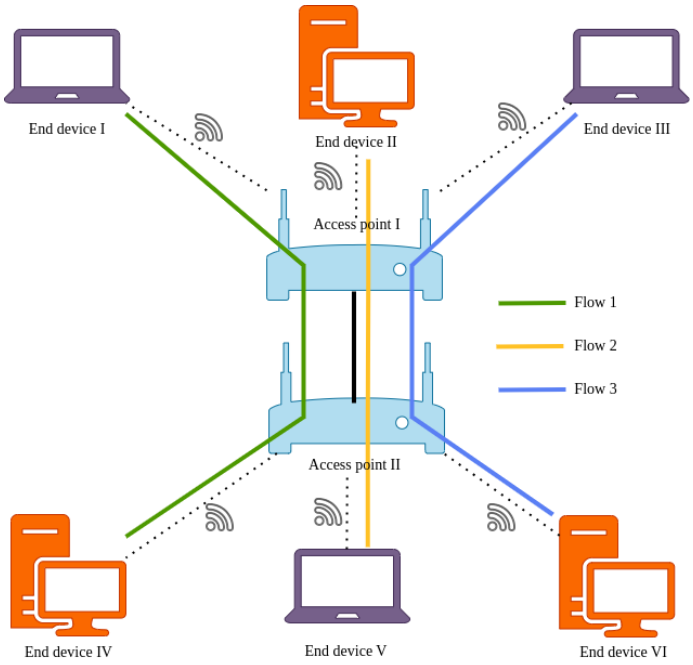


Fig. 6. Multi-flow wireless network setup in IDLab Testbed.

The designed NACC (explained in Section VI-B) utilizes this updated capacity information to modify the data transfer rate and adapt to other flows in the network.

B. Making the Network-aware Congestion Control Algorithm Application-aware

The network-aware congestion control algorithm is still not aware of the APP-REQ and uses only the real-time network context. With APP-REQ processing in the intermediate nodes, the source nodes now have distributed knowledge of other flows in the network and its calculated capacity. We will make a few modifications to the network-aware congestion control algorithm mentioned in Section V and make it application-aware as well.

No FC monitoring: The intermediate node itself takes into account the other flows in the network and distributes the capacity accordingly. Therefore, the source node does not have to consider FC, and B_CWND is not set based on FC.

Modified CWND bounds: The CWND_L guarantees the minimum number of packets sent and CWND_UP limits the maximum number of packets sent. Therefore, to provide a guaranteed service to an application, the PRIORITY of the application is used to set the CWND_L (9). To maintain fairness among different flows, the application should not exceed its data transfer rate above the CAPACITY, hence the CAPACITY is used to set the CWND_UP (10). The CWND_UP is calculated using 16% of DR (16% is the combined overhead by lower layers and 20% margin as mentioned previously), RTT, and MSS (MSS is multiplied by 8 to convert bytes to bits), along with the CAPACITY_r obtained from the intermediate node, which acts as an upper threshold and takes into account the percentage of DR available for the node. The CWND_L and CWND_UP of application- and network-aware congestion control algorithm are directly proportional to PRIORITY and CAPACITY, respectively.

$$CWND\_L = 10 \cdot PRIORITY_r \tag{9}$$

$$CWND\_UP = CWND\_L + \frac{CAPACITY_r \cdot (DR \cdot 0.16) \cdot RTT}{8 \cdot MSS} \tag{10}$$

Partiality among the flows: The applications with higher PRIORITY are given more preference as compared to the lower PRIORITY ones. Hence the algorithm is designed to increase the CWND aggressively for a flow with higher PRIORITY. Similarly in the presence of a higher PRIORITY application, the algorithm is designed to decrease the CWND aggressively for a flow with lower PRIORITY. This is achieved by modifying the increase rate of CWND and making it proportional to PRIORITY along with Q and A (11) and modifying the decrease rate of CWND on packet loss and making it inversely proportional to PRIORITY (12).

These simple modifications to the network-aware congestion control algorithm make the algorithm application-aware. In this article, we have designed a NACC with the option to switch to only a network-aware congestion control algorithm. Though we utilize the real-time network parameters and APP-REQ of other flows in the network, there are still certain parameters that are not accessible to the transport layer and can result in packet loss. Hence we have designed the algorithm such that it takes into account these packet losses and still maintains a stable data transfer. The APP-REQ processing in intermediate nodes is implemented using the Click modular

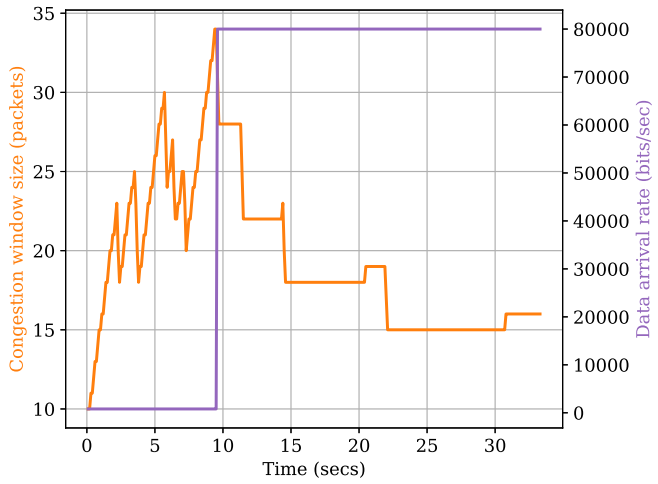


Fig. 7. Responsiveness of CWND of NACC to data arrival rate.

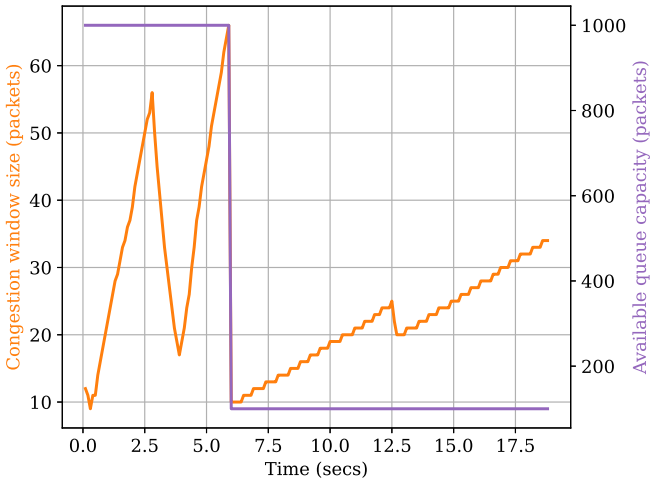


Fig. 8. Responsiveness of CWND of NACC to available queue capacity.

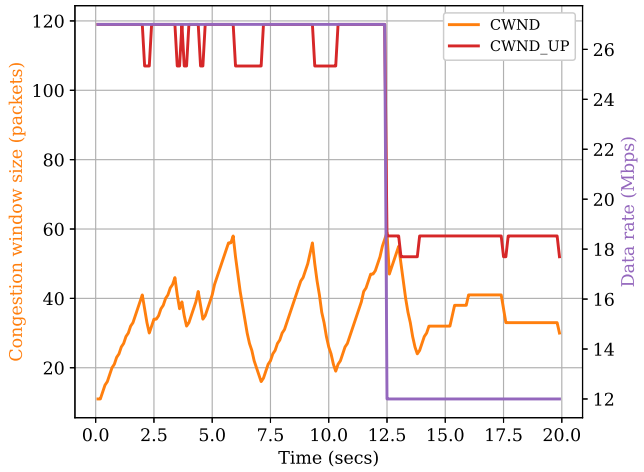


Fig. 9. Responsiveness of CWND of NACC to data rate of the wireless channel.

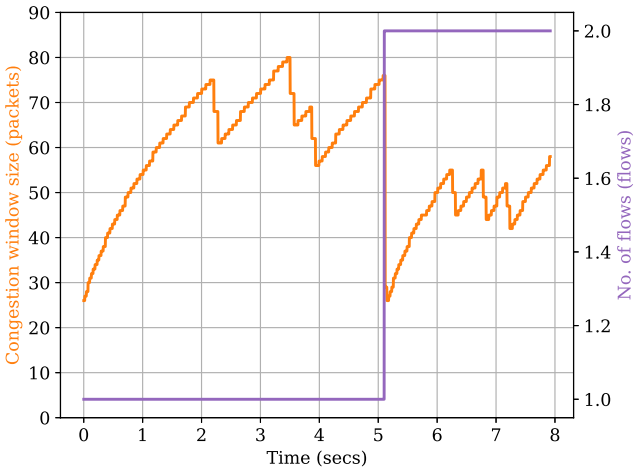


Fig. 10. Responsiveness of CWND of NACC to number of flows in the network.

router framework[6]. The modifications in the congestion control algorithm are made in the previous implementation of the network-aware congestion control algorithm using CCP.

$$CWND = B\_CWND + \frac{PRIORITY_r \cdot Q}{A \cdot 8 \cdot MSS} \tag{11}$$

$$CWND = B\_CWND \cdot (1 - 0.1 \tag{12}$$

$$\times (MAX\_PRIORITY - PRIORITY_r)) \tag{13}$$

VII. Results and Discussion

A. Responsiveness of the NACC

As discussed in Section V-A, key parameters such as Q, A, FC, and DR, have a direct impact on the congestion. The increase or decrease in any of these parameters should trigger a positive or negative change in the behavior of the NACC. Hence we tested the impact of these parameters on the designed congestion control algorithm. A network with two end devices and a black box was set up in Mininet-wifi[7] to test the responsiveness of the designed algorithm (Fig. 4). The black box was implemented using the Click modular router, to encapsulate dummy network parameters in INT. The responsiveness was tested for changes in Q, A, FC, and DR of the network.

With a sudden increase in A in Fig. 7, the congestion control algorithm drastically reduces the CWND, thus the data transfer rate adapts to the increase in the load. Similarly, when the available queue capacity at the access point suddenly decreases in Fig. 8, the CWND also decreases. When the value of the DR at the access point decreases as shown in Fig. 9, the CWND_UP is reset according to this change and

[6]https://github.com/kohler/click

[7]https://mn-wifi.readthedocs.io/en/latest/

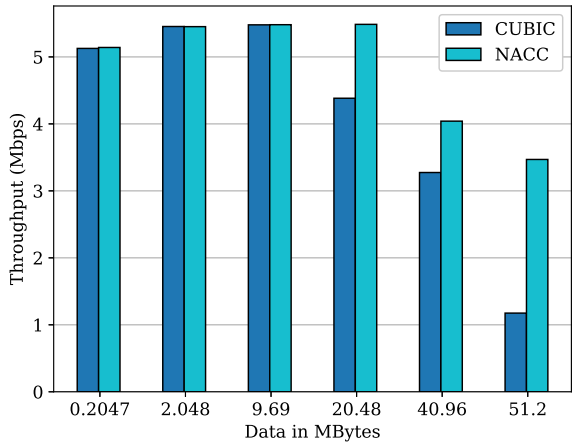


Fig. 11. Throughput comparison of CUBIC and NACC.

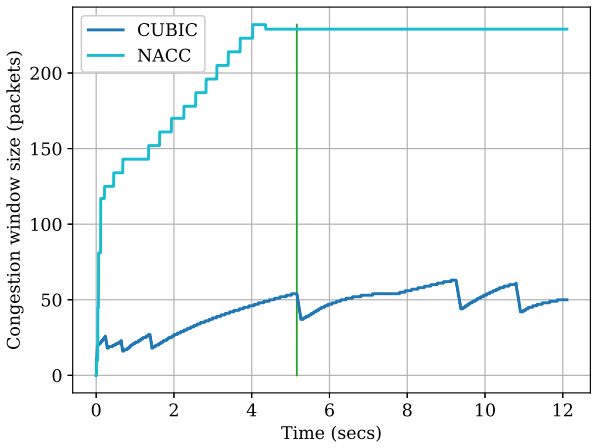


Fig. 12. CWND over time of CUBIC and NACC.

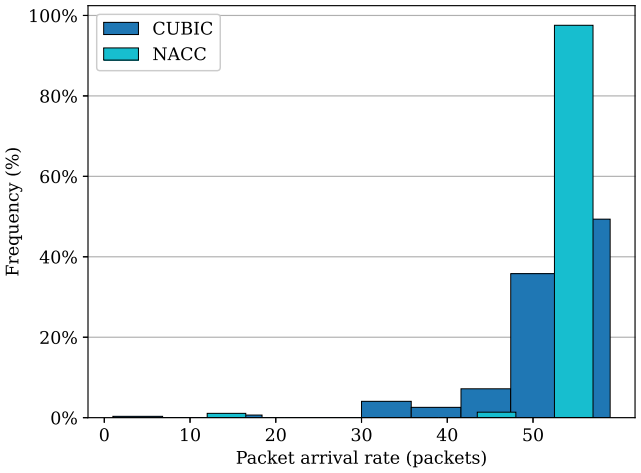


Fig. 13. Frequency distribution of CWND of CUBIC and NACC.

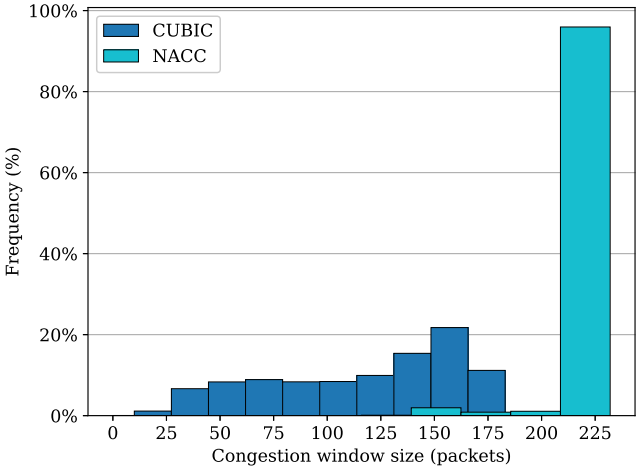


Fig. 14. Frequency distribution of load at the intermediate node generated by CUBIC and NACC.

thus the designed algorithm also reduces the data transfer rate by lowering CWND. Also when the number of flows in the network increases, the congestion control algorithm is notified by INT through FC, and thus the algorithm reduces the CWND as shown in Fig. 10. Therefore we can conclude that the designed congestion control algorithm is highly responsive to the changes in the network.

B. Benchmarking Against the CUBIC Congestion Control Algorithm

The authors in [14] and [15], have proven the CUBIC congestion control algorithm to be better than other congestion control algorithms like TCP Tahoe, TCP Reno, including binary increase congestion control (BIC-TCP) based on which the CUBIC algorithm was built. The CUBIC algorithm has been used in Linux kernels since version 2.6 [16]. Hence we have benchmarked the designed NACC algorithm against the CUBIC congestion control algorithm. The performance was compared in terms of throughput, stability of the CWND and A, and progression of the CWND over time. The tests were performed on a network emulated in Mininet-wifi with two end devices and an access point as show in Fig. 5.

Fig. 11 indicates that the throughput of the designed congestion control algorithm is better than the CUBIC. Especially with the increase in the load (bytes sent), the throughput of the designed congestion control algorithm is almost three times better than the CUBIC. On starting the data transfer, the designed congestion control algorithm tries to achieve a stable CWND as soon as possible. It also tries to maintain the same CWND (or at least vary in a small window) until all the data is transferred. The designed congestion control algorithm reaches the maximum possible CWND for the given network before the CUBIC algorithm as indicated by the green line in Fig. 12. From the frequency distribution Fig. 13, we see that the CWND of CUBIC is highly variable whereas the CWND of the designed congestion control algorithm is more stable and tries to maintain the same value. The impact of this can be seen on the frequency distribution of A in Fig. 14.

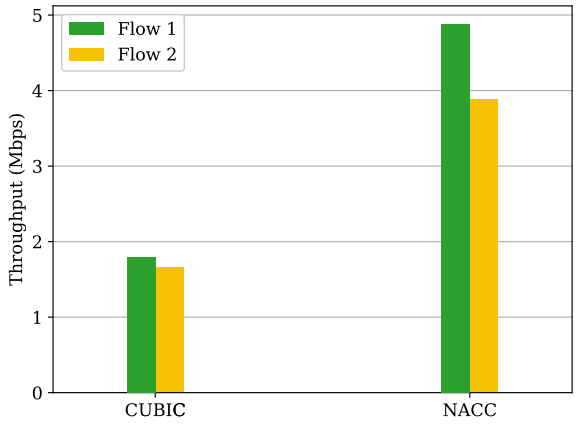


Fig. 15. Throughput comparison of CUBIC and NACC for flow 1 and 2 with priorities 4 and 2, respectively.

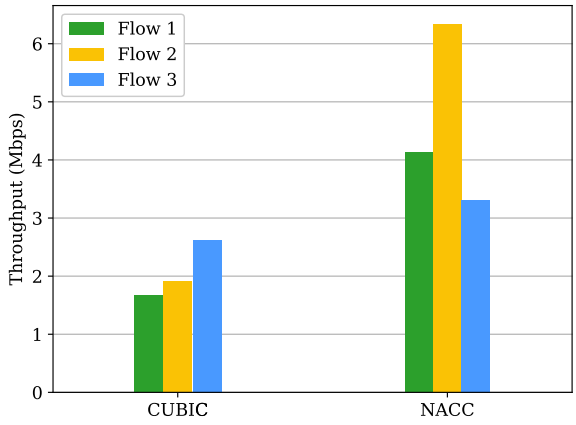


Fig. 16. Throughput comparison of CUBIC and NACC for flow 1, 2, and 3 with priorities 2, 4 and 1, respectively.

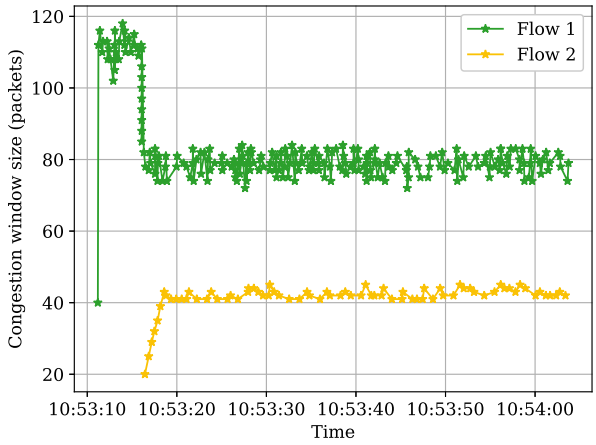


Fig. 17. CWND over time of NACC for flow 1 and 2.

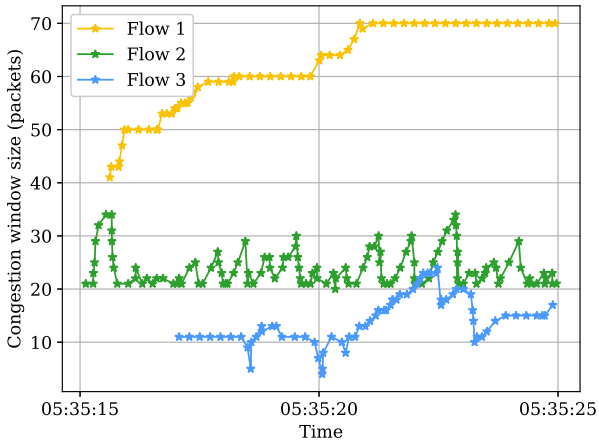


Fig. 18. CWND over time of NACC for flow 1, 2, and 3.

C. Bandwidth Fairness Based on Application Requirements

The designed NACC distributes the capacity based on the APP-REQ as opposed to the CUBIC which achieves equal fairness irrespective of the application priorities. Therefore, the performance of the designed congestion control algorithm was evaluated in a multi-flow wireless network in IDLab testbed[8]. The testbed setup consisted of two access points connected by a wired connection and six end devices, a set of three connected to an access point wirelessly as shown in Fig. 6.

For the purpose of evaluation, the application could specify priority levels between 4 to 1, 4 being the highest and 1 being the lowest priority (Listing V-B). The throughput of two flows with different priorities (flow 1 with priority 4 and flow 2 with priority 2) was measured for CUBIC and the designed congestion control algorithm is plotted in Fig. 15. From the plot, it can be seen that the throughput of flow 1 is the same as flow 2 for CUBIC, whereas, it is greater than flow 2 when the NACC is used. The overall throughput achieved by the new algorithm is twice the throughput achieved by CUBIC. The designed congestion control algorithm sets the CWND_L and CWND_UP based on the APP-REQ, hence when the second flow starts, the access point notifies the first flow of the new capacity distribution, and the CWND bounds are reset. Therefore flow 1 reduces its CWND, but there is still a marginal gap between the CWND of the two flows as shown in Fig. 17. This achieves differentiated service based on the priorities of the flows. Similarly, the algorithm was tested for three flows with different priorities (flow 1 with priority 2, flow 2 with priority 4, and flow 3 with priority 1) and the throughput was measured. The results in Fig. 16 indicate that on average the overall throughput of the NACC algorithm is twice the throughput of CUBIC. We can see the difference in the throughput of each flow based on their priority which is also reflected in the differences of the CWND of the three flows in Fig. 18. We can observe that flow 2 and flow 1 have a priority difference of two, hence a larger gap in CWND as

[8]https://doc.ilabt.imec.be/ilabt/wilab/

compared to flow 1 and flow 3 with a priority difference of only one.

VIII. Conclusion and Future Work

Recent innovations like INT and APP-NET integration have enabled the possibility of applications being aware of the real-time network context and network being aware of APP-REQ. Utilizing these features, we have designed a rule-based network- and application-aware adaptive congestion control algorithm, which operates based on the real-time network context and distributed knowledge on aggregated flow information. The algorithm takes into account the APP-REQ, and its priorities and achieves service differentiation among different flows in a multi-flow network architecture. It provides a congestion free data transfer service in a wireless distributed network architecture. The performance of the designed algorithm was evaluated by benchmarking with the CUBIC congestion control algorithm. The designed algorithm achieved three times more throughput than the CUBIC when 50 MB of data was sent. The designed algorithm distributed the capacity among different flows based on their priority and achieved a throughput that is twice as CUBIC in a multi-flow wireless network.

With several diverse applications with different priorities being introduced everyday, the designed congestion control algorithm is capable of addressing these requirements and achieving differentiation based on their priorities in a multi-flow distributed network. It paves the way for an adaptive transport and application layer protocol for private professional networks and can complement more centralized scheduling mechanisms. As a next step, we will look into adapting other mechanisms of a transport layer protocol and in better taking the dynamics of wireless settings into account. We will also look into designing adaptive application layer protocol for private professional networks.

References

[1] Cisco. “Cisco annual Internet report (2018–2023) white paper." 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] A. Langley *et al.*, “The quic transport protocol: Design and internet-scale deployment," in *Proc. ACM SIGCOMM*, 2017.

[3] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke, “Tighter application-network interfacing to drive innovation in networked systems," in *Proc. ACM SIGCOMM*, 2021.

[4] J. Haxhibeqiri, P. H. Isolani, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke, “In-band network monitoring technique to support SDN-based wireless networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 1, Mar. 2021.

[5] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, “Adaptive transport layer protocols using in-band network telemetry and eBPF," in *Proc. IEEE WiMob*, 2021.

[6] J. Zhang *et al.*, “Sextant: Enabling automated network-aware application optimization in carrier networks," in *Proc. IFIP/IEEE IM*, 2021.

[7] M. Maris, T. Halpin, D. Ezeh, K. Miu, and J. de Oliveira, “Adaptive packet transmission in response to anomaly detection in software defined smart meter networks," 2021, *arXiv:2112.04602*.

[8] T. Kimura, T. Kimura, A. Matsumoto, and K. Yamagishi, “Balancing quality of experience and traffic volume in adaptive bitrate streaming," *IEEE Access*, vol 9, pp. 15530–15547, Jan. 2021.

[9] T. Krüger, D. Hausheer, “Towards an API for the path-aware Internet," in *Proc. ACM SIGCOMM*, 2021.

[10] J. Liu, S. Lu, and Q. Yang, “Accurate-ECN: An ECN enhancement with inband network telemetry," in *Proc. IEEE LCN*, 2022.

[11] Y. Ma *et al.*, “ Multi-objective congestion control," in *Proc. ACM EuroSys*, 2022.

[12] L. P. Verma, V. K. Sharma, M. Kumar, and D. Kanellopoulos, “A novel delay-based adaptive congestion control TCP variant," *Comput. Electr. Eng.*, vol. 101, Jul. 2022.

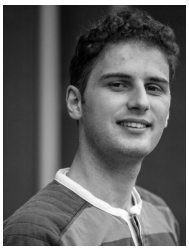
[13] A. Narayan *et al.*, “Restructuring endpoint congestion control," in *Proc. ACM SIGCOMM*, 2018.

[14] S. Ha, I. Rhee, and L. Xu, “CUBIC: A new TCP-friendly high-speed TCP variant," *ACM Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[15] T. Kozu, Y. Akiyama, and S. Yamaguchi, “Improving RTT fairness on CUBIC TCP," in *Proc. IEEE CANDAR*, 2013.

[16] L. Xu *et al.* “Cubic for fast long-distance networks," Internet Eng. Task Force, Fremont, CA, USA, RFC 8312, 2018.

Ramyashree Venkatesh Bhat received her M.Sc. (2020) and B.E. (2017) degrees in Communication Systems from Technical University of Munich, Munich, Germany and Electronics and Communications Engineering from PES Institute of Technology (now PES University), Bengaluru, India respectively. She is currently pursuing her Ph.D. degree in Engineering Computer Science from Ghent University, Ghent, Belgium.



Jetmir Haxhibeqiri received the Master’s degree in Engineering, Information Technology, and Computer Engineering from RWTH Aachen University, Aachen, Germany, in 2013, and the Ph.D. degree in Engineering Computer Science from Ghent University, Ghent, Belgium, in 2019. He is currently a Senior Researcher with the Internet Technology and Data Science Lab, Ghent University and IMEC, Leuven, Belgium. His research interests include wireless communications technologies (IEEE 802.11, IEEE 802.15.4e, LoRa) and their application, IoT, wireless time-sensitive networking, in-band network monitoring, and wireless network management.

Ingrid Moerman received a degree in Electrical engineering in 1987, and a Ph.D. degree from Ghent University in 1992, where she became a part-time Professor in 2000. She is a Staff Member with IDLab, Core Research Group, imec with research activities embedded with Ghent University and the University of Antwerp. Her main research interests include cooperative and intelligent radio networks, real- time software-defined radio, time-sensitive networks, dynamic spectrum sharing, co-existence across heterogeneous wireless networks, vehicular networks, open-source prototyping platforms, software tools for programmable networks, next-generation wireless networks (5G/6G), and experimentally supported the research.

Jeroen Hoebeke is an Associate Professor in the Internet Technology and Data Science Lab of Ghent University and imec. He is conducting and coordinating research on wireless (IoT) connectivity, embedded communication stacks, deterministic wireless communication and wireless network management. This expertise has been applied in a variety of application domains such as logistics, Industry 4.0, building automation, healthcare and animal monitoring. He is particularly active in national funded projects as well as in defining, executing and managing such projects. He has also been involved in several EU research funded projects and is author or co-author of more than 150 publications in international journals or conference proceedings.