FACULTY OF ENGINEERING AND ARCHITECTURE

Next-Generation Adaptive Transport Layer Protocols Based on Network Context and Distributed Knowledge

Ramyashree Venkatesh Bhat

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors
Prof. Jeroen Hoebeke, PhD - Prof. Ingrid Moerman, PhD
Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

February 2025

GHENT UNIVERSITY

ISBN 978-94-6355-954-6
NUR 986, 983
Wettelijk depot: D/2025/10.500/14

Members of the Examination Board

Chair

Prof. Em. Daniël De Zutter, PhD, Ghent University

Other members entitled to vote

Prof. Olivier Bonaventure, PhD, Université catholique de Louvain
Prof. Jeroen Famaey, PhD, Universiteit Antwerpen
Prof. Adnan Shahid, PhD, Ghent University
Tim Wauters, PhD, Ghent University

Supervisors

Prof. Jeroen Hoebeke, PhD, Ghent University
Prof. Ingrid Moerman, PhD, Ghent University

Acknowledgment

"If I have seen further, it is by standing on the shoulders of giants."

– Sir. Isaac Newton

I would like to express my deepest gratitude to my advisors, Prof. Ingrid Moerman and Prof. Jeroen Hoebeke, for their continuous support, patience, and guidance throughout my Ph.D. journey. Their insightful feedback and encouragement have been invaluable. I would like to thank my colleague (but not my boss/supervisor) Jetmir, for always being accessible, receptive of ideas and for asking critical questions when the things were half-baked. Our interactions and brain-storming sessions with Prof. Hoebeke are the corner stone of where I stand today.

I am also grateful to the members of my dissertation committee, Prof. Olivier Bonaventure, Prof. Jeroen Famaey, Prof. Adnan Shahid, Dr. Tim Wauters, and Prof. Em. Daniel De Zutter, for their time, valuable comments, and suggestions that greatly improved the quality of my thesis work.

Pieter and Vincent, thanks for responding to my SOS calls when there were problems with the wireless testbed. Your work has been an enabler for my research, and I am grateful for having access to this state-of-the-art facility.

Special thanks to my colleagues turned friends, with whom I have shared many joyous moments, right from discussions in our office, the healthy diet debates during lunch and finally huff and puff our way to the 11th floor by stairs. Pablo, Mohammad, Vasilis, Dries, Andy and Ozgur, I will miss seeing you all on an almost daily basis! I am also thankful to my other current and ex-colleagues, Merkebu, Amina, Ben, Wei, Xianjun, Mathias, Ihtisham, Abderraouf, Hojjat, and Prof. Eli De Poorter. Thank you for your support, camaraderie, and for making this journey enjoyable. Your encouragement and your own achievements have been a great source of motivation and pride for me.

I would also like to acknowledge the financial support provided by UGent and SBO VERI-END project, which made this research possible.

Finally, I am deeply indebted to my family, especially my parents, Suma and Venkatesh; my partner, Sanketh; and my sister, Chaitra; for their unwavering love, support, and sacrifices. Special mention to my in-laws for their encouragement.

This accomplishment would not have been possible without you all.

Gent, February 2025
Ramya

Table of Contents

List of Figures

List of Tables

List of Acronyms

A

AR	Augmented Reality
ACK	Acknowledgment
APP-NET	Application-Network interaction
AP	Access Point
API	Application Programming Interface
ANA	Application Network Agent
AI	Artificial Intelligence

B

BSD	Berkeley Software Distribution
BBR	Bottleneck Bandwidth and Round-trip Propagation Time

C

CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CC	Congestion Control
CCP	Congestion Control Plane
COTS	Commercially Off The Shelf
CoAP	Constrained Application Protocol
CMDP	Context-aware Markov Decision Process

CBOR Concise Binary Object Representation

D

DiffServ Differentiated Services

E

eBPF extended-Berkeley Packet Filter
ECN Explicit Congestion Notification

F

FCCC Feedback-based Control loop Congestion Control

G

Gbps Giga bit per second

H

HTTP Hyper Text Transport Protocol

I

IP Internet Protocol

IEEE Institute of Electrical and Electronics Engineers
INT In-band Network Telemetry
IPv6 Internet Protocol version 6
IoT Internet of Things

K

kbps kilobit per second
KB Kilo Bytes

L

LMR Land Mobile Radio
LAN Local Area Network

M

MSS Maximum Segment Size
Mbps Mega bit per second
MAC Medium Access Control
MB Mega Bytes
MDP Markov Decision Process
MAP-co Multi-AP coordination
MLO Multi-link operation

N

NACC Network- and Application-aware adaptive Congestion Control

O

OSI Open Systems Interconnection

OFDMA Orthogonal Frequency Divison Multiple Access

P

PI Proportional-Integral

P4 Programming Protocol-independent Packet Processors

Q

QoS Quality of Service

QUIC Quick UDP Internet Connections

R

RTT Round Trip Time

RSSI Received Signal Strength

RL Reinforcement Learning

RACC Reactive Airtime feedback-based Congesiton Control

R-TWT Restricted Target Wake Time

S

SACK Selective Acknowledgment

SNR Signal to Noise Ratio
SYN Synchronize
SYN-ACK Synchronize-Acknowledgment

T

TCP Transmission Control Protocol
TCP/IP Transmission Control Protocol/Internet Protocol
TXOP Transmit Opportunity
TSN Time-sensitive networking

U

UDP User Datagram Protocol

V

VR Virtual Reality

W

Wi-Fi Wireless Fidelity

List of Symbols and their units

$cwnd$	congestion window size (packets)
$ssthresh$	slow start threshold (packets)
W_max	$cwnd$ during loss event in CUBIC algorithm (packets)
q	available queue capacity (packets/second)
a_{packet}	packet arrival rate (packets/second)
fc	number of flows passing through the intermediate network node (no. of flows)
β	$cwnd$ decrease constant, i.e. $cwnd$ is decreased by $1 - \beta$ value
a_{data}	Data arrival rate (bits/second)
$loss_w$	Packet loss due to the quality of wireless medium (packets)
dr	Data rate of the link (Mbps)
rtt	Round trip time (second)
MSS	Maximum Segment Size (bytes)
$cwnd_l$	$cwnd$ does not go below this lower bound (packets)
$cwnd_{up}$	$cwnd$ does not exceed this upper bound (packets)
a_{up}	Data arrival rate does not exceed this upper bound (bits/second)
$cwnd_{prev}$	$cwnd$ of previous data transfer (packets)
a_{diff}	Rate of increase in the load at the network node (percentage)
$priority$	Required application priority (-)
$max_{priority}$	Total no. of available priority levels (-)
$capacity$	Percentage of dr available for data transfer (percentage)
dc	Device count, Number of end devices using the channel (none)
ts	Timeslot, time period over which the AP calculates the aggregated airtime usage (second)
t_p	Airtime of packet (second)
p_{ip}	IP payload (byte)
sp	Percentage of ts that can be used by the device including the margin for new connection initiation (%)

ar	Aggregated airtime feedback calculated by AP over ts (%)
w_ar	Weighted average of aggregated airtime feedback calculated by AP over ts (%)
$loss_w$	Wireless packet loss (packet)
$ertt$	RTT estimated by AP (second)
$e2a_l$	Latency between end device to AP (second)
t_p	Airtime of packet (second)
c_p	Correction factor (none)
$error$	Error between the aggregated airtime feedback and target sp (none)
K_p	Proportional constant of PI controller (none)
K_i	Integral constant of PI controller (none)
e_p	Proportional error of PI controller (none)
e_i	Integral error of PI controller (none)
PI_{error}	The summation of proportional and integral error in PI controller equation (none)
$loss_p$	Packets lost due to congestion in the network (packet)
$bytes_{ACK}$	Bytes acknowledged by the receiver (byte)
$bytes_{SACK}$	Bytes selectively acknowledged by the receiver (byte)
p_{size}	Size of the packet (byte)
$flow_{timeout}$	Flow timeout notification (boolean)

Samenvatting
– Summary in Dutch –

Connectiviteit is het belangrijkste hulpmiddel in de wereld van vandaag. Vooral tijdens de COVID-pandemie heeft connectiviteit een cruciale rol gespeeld bij het draaiende houden van kritieke communicatie. Connectiviteit is ingebed in verschillende sectoren zoals industrie, productie, gezondheidszorg, financiën, leger, enz. door de brede acceptatie van particuliere professionele draadloze netwerken. Dit heeft het aantal apparaten dat op het netwerk is aangesloten enorm vergroot. Daarnaast hebben de diverse toepassingen zoals cloud computing, AR/VR, alomtegenwoordige netwerken, interactie tussen mens en machine, enz. in deze netwerken uiteenlopende eisen op het gebied van doorvoer en latentie. Bovendien hebben een paar toepassingen ook tijdkritische vereisten, die tijdgevoelig en deterministisch moeten zijn. De impact van deze gebeurtenissen heeft geresulteerd in een groot aantal heterogene verkeersstromen in het netwerk. De belofte van een hoge gegevensdoorvoer en lagere latenties in de recente generaties Wi-Fi-standaarden heeft het gebruik ervan doen toenemen en het is een van de wijdverbreide communicatietechnologieën in particuliere professionele netwerken en voor tijdgevoelige netwerken. De strenge eisen van de privénetwerken en tijdgevoelige netwerken worden nu versterkt door de uitdagingen die draadloze technologieën met zich meebrengen.

Het bekabelde domein heeft Software-Defined Networking (SDN) aangenomen om het besturings- en datavlak te scheiden, waardoor herconfiguratie van het netwerk op afstand mogelijk is. P4-programmering wordt gebruikt voor flexibiliteit in het forwardingvlak. De uitbreiding van SDN naar het draadloze domein heeft geleid tot frameworks voor datavlakprogrammeerbaarheid in Wi-Fi en 5G. Het hebben van dergelijke datavlakprogrammeerbaarheid en nieuwe tijdgevoelige netwerkfacilitators kan helpen om determinisme te realiseren in gemengde bekabelde en draadloze netwerken, wat de weg vrijmaakt voor servicedifferentiatie en prioritering van applicaties. Het begrijpen van de status en applicatiebehoeften van het netwerk is echter cruciaal om programmeerbaarheid volledig te benutten. Ondanks het brede scala aan applicaties, zijn ze beperkt tot het kiezen van een beperkt aantal quality-of-service-opties. De forwardingprioriteiten die worden geboden door de bestaande netwerkcomponenten zijn statisch van aard en bieden niet voldoende granulariteit. Om de configureerbaarheid en flexibiliteit in de netwerken mogelijk te maken en het netwerk aan te passen en te herconfigureren aan de applicatievereisten, is het daarom essentieel om de applicatielaag te betrekken bij het beheer van het netwerk

door de vereisten ervan te erkennen aan de netwerklaag. De APP-NET-interface stelt applicaties in staat om hun verkeers- en monitoringvereisten te specificeren aan de netwerklaag in draadloos-bekabelde systemen die kunnen worden gebruikt om het onderliggende draadloze netwerk te configureren. Om de netwerkstatus in realtime te begrijpen, is het essentieel om de netwerkprestaties continu te bewaken en te verifiëren. De beperkingen van out-of-band-methoden voor meting en evaluatie hebben geleid tot een grotere focus op in-band netwerktelemetrie (INT). Oorspronkelijk beperkt tot geschakelde bekabelde netwerken, is het idee nu ook toegepast op draadloze systemen. Het INT-framework voor draadloze netwerken heeft bewezen een lage overhead, fijnmazig en betrouwbaar te zijn. INT kan end-to-end en hop-by-hop-knooppuntinformatie verzamelen, samen met realtime draadloze linkinformatie.

Voldoende informatie over de applicatievereisten en netwerkconditie kan worden verzameld en gebruikt met data plane-programmeerbaarheid, applicatie-netwerk-interactie en geavanceerde telemetrie zoals INT. De bestaande tijdsgevoelige plannings- en slicingalgoritmen kunnen echter alleen inspelen op de tijdskritische stromen. Wanneer het aantal stromen in het netwerk honderden of zelfs duizenden nadert, wordt het voor tussenliggende netwerkknooppunten moeilijk om een nauwkeurig afgestemde, gedifferentieerde behandeling voor elke stroom te handhaven en tegelijkertijd meer functionaliteit (zoals verkeersvorming, wachtrijen, tijdsgevoeligheid, enz.) te accommoderen, waardoor uiteindelijk hun capaciteit om extra functies te ondersteunen wordt overschreden.

Als zodanig nemen de rekenkracht en het geheugen van eindapparaten toe, en sommige van hen hebben nu de capaciteit voor analyses en intelligente besluitvorming met behulp van kunstmatige intelligentie. Gezien de bovengenoemde beperkingen op de fijnmazige verwerking van stromen en de groeiende mogelijkheden van eindapparaten, is het passend om te vragen hoe en in welke mate eindapparaten, met name hun transportlaagprotocollen, kunnen samenwerken om gezamenlijk een optimale netwerkfunctie te bereiken.

De transportlaag van het OSI-model biedt logische host-to-hostcommunicatie. In tegenstelling tot UDP biedt TCP een betrouwbare, foutloze, geordende, gegarandeerde gegevensoverdrachtsservice met behulp van congestie- en stroomcontrolemechanismen. Verschillende onderzoeken hebben zich gericht op het verbeteren van congestiecontrole-algoritmen voor betere netwerkprestaties. Traditionele congestiecontrole-algoritmen detecteren problemen zoals pakketverlies, verhoogde vertraging of gebruiken hybride benaderingen op basis van beperkte end-to-end-informatie. Om latentie te verminderen en de netwerkcapaciteit te optimaliseren, worden nieuwe feedbacktechnieken gebruikt. Het QUIC-protocol van Google gebruikt bijvoorbeeld spin-bit-pakketten om congestie te detecteren. Terwijl sommige congestiecontrole-algoritmen de expliciete congestiemelding (ECN)-bit in de IP-header gebruiken. ECN heeft echter beperkingen, waaronder vertraagde meldingen en incompatibiliteit met switches. Het nieuwe L4S-protocol herinterpreteert ECN voor prestaties met lage latentie, maar vereist speciale routerwachtrijen en kan conflicteren met traditionele congestiecontrole-algoritmen. Bovendien zijn de bestaande congestiecontrole-algoritmen ontworpen om de hoge bandbreedte van

bekabelde netwerken te benutten en hun doorvoer te maximaliseren. In het geval van draadloos zijn er andere uitdagingen zoals verliesgevend medium, interferentie, veranderende kanaalomstandigheden, enz. De laatste tijd is er meer belangstelling voor op leren gebaseerde congestiecontrole-algoritmen. Maar deze algoritmen gebruiken nog steeds geen netwerkfeedback of gebruiken beperkte feedback via ECN. Bovendien worden deze algoritmen offline getraind, wat in tegenspraak is met de verandering en onvoorspelbaarheid van draadloze verbindingen.

Dit proefschrift behandelt de bovenstaande kwesties vanuit het perspectief van netwerkprotocollen op hogere lagen door ze te betrekken bij de besluitvorming over het netwerk. Met de recente innovaties in draadloze netwerken zoals netwerktelemetrie, applicatie-netwerkinteractie, dataprogrammeerbaarheid van de tussenliggende netwerkknooppunten en programmeerbaarheid van de kernel via eBPF, stelt dit proefschrift adaptieve congestiecontrole-algoritmen van de volgende generatie voor voor particuliere professionele netwerken. In eerste instantie wordt de haalbaarheid van adaptieve transportlaagprotocollen onderzocht door de relatie te vinden tussen realtime netwerkparameters van INT en de grootte van het congestievenster. eBPF is geïntegreerd met INT om TCP-gegevens te synchroniseren met INT-informatie. De resultaten van deze studie gaven aan dat het gedrag van de grootte van het congestievenster direct invloed heeft op de belangrijkste parameters van tussenliggende netwerkknooppunten, zoals de pakketaankomstsnelheid, de bufferwachtrijcapaciteit en het aantal stromen. Met behulp van de INT-informatie werd het bestaande CUBIC-congestiecontrolealgoritme aangepast om pakketverlies als gevolg van de draadloze verbindingsomstandigheden te overwegen. De verbeterde prestaties van het algoritme valideerden het potentieel van gedetailleerd inzicht in de netwerkstatus.

Door de relatie tussen congestiecontrolealgoritmen en netwerkcontext verder te benutten, is een netwerk- en applicatiebewust congestiecontrolealgoritme ontworpen. Door gebruik te maken van de dataprogrammeerbaarheid van het doorstuurvlak, is het tussenliggende knooppunt geconfigureerd om de applicatieaanvraag te verwerken (gefaciliteerd door APP-NET en INT) en capaciteit en prioriteit toe te wijzen aan elk eindapparaat in het netwerk. Deze toewijzing wordt verder gecommuniceerd naar het congestiecontrolealgoritme van het eindapparaat en het algoritme wijzigt de gegevensoverdrachtssnelheid op basis van deze feedback en de realtime netwerkcontext. Daarom is het ontworpen algoritme op de hoogte van zowel applicatievereisten als veranderende netwerkstatussen. Naast congestievrije service luistert het algoritme naar de behoeften van de applicatie en biedt het fijnmazige servicedifferentiatie, waardoor een deel van de last van tussenliggende knooppunten wordt overgenomen. Het algoritme presteert ook beter dan het bestaande CUBIC-congestiecontrolealgoritme.

Om het probleem van de degradatie van de doorvoer in Wi-Fi-netwerken aan te pakken door een eerlijk kanaaltoegangsmechanisme en om airtime-eerlijkheid te bereiken, is een reactief en adaptief congestiecontrole-algoritme ontworpen. Het toegangspunt heeft de airtime expliciet toegewezen aan elk eindapparaat, uitsluitend op basis van het aantal eindapparaten in het netwerk. Hiermee paste het algoritme de gegevensoverdrachtssnelheid van de toepassing aan. De reactieve aard van

het congestiecontrole-algoritme resulteerde in onderbenutting van het draadloze netwerkkanaal. Om dit te beperken, is een proactief congestiecontrole-algoritme ontworpen.

Dit algoritme is een feedback-gebaseerd proportioneel-integraal (PI) controle-lus congestiecontrole-algoritme. Het tussenliggende knooppunt is geconfigureerd om het airtime-gebruik van elk eindapparaat bij te houden en de geaggregeerde airtime-informatie te communiceren. Het ontworpen congestiecontrole-algoritme gebruikt deze informatie om de fout in airtime-gebruik te berekenen, waardoor het aantal te delen datapakketten tijdens de volgende gegevensoverdracht wordt gecorrigeerd. In tegenstelling tot de bestaande algoritmen voor congestiecontrole zorgt het ontworpen algoritme voor eerlijke zendtijd en is het bestand tegen veranderende kanaalkwaliteit en andere netwerkparameters door de gegevensoverdrachtssnelheid van de toepassingen aan te passen.

Hoewel de ontworpen algoritmen aanpasbaar zijn aan de draadloze netwerkomstandigheden, is het gunstig om het toekomstige netwerk te kunnen anticiperen. Het op PI-regelkringen gebaseerde congestiecontrolealgoritme is al proactief van aard, maar om de mogelijkheden verder uit te breiden naar toepassingsvereisten en de leermethoden te verkennen, werd een op reinforcement learning gebaseerde benadering onderzocht. Omdat de draadloze netwerkparameters zoals kanaalkwaliteit, andere concurrerende knooppunten en het aantal stromen, onafhankelijk van de acties die door een eindapparaat worden ondernomen, een aanzienlijke invloed hebben op de algehele doorvoer en latentie van het eindapparaat, wordt de op multi-context-bewuste RL gebaseerde congestiecontrole gemodelleerd als een contextueel Markov Decision Process (CMDP). Er worden drie verschillende op multi-context-bewuste RL gebaseerde congestiecontrolealgoritmen geïntroduceerd die het aantal toepassingsgegevenspakketten dat naar het netwerk wordt verzonden, aanpassen op basis van toepassingsvereisten en realtime netwerkinformatie. Twee van deze oplossingen werden geïmplementeerd en getest in het wifi-netwerk.

Alle ontworpen algoritmen voor congestiecontrole zijn geïmplementeerd op commercieel beschikbare draadloze apparaten en de validatie tegen bestaande algoritmen voor congestiecontrole werd uitgevoerd in de WiLab2-testbed van IDLab. De betere prestaties van ontworpen algoritmen in termen van servicekwaliteit, airtime-eerlijkheid en aanpasbaarheid aan draadloze omstandigheden verzekeren het belang van realtime netwerkkennis in protocollen op hogere lagen. Naast het leveren van congestievrije services, faciliteren de ontworpen algoritmen airtime-eerlijkheid, toepassingsvereisten en passen ze zich aan veranderende netwerkomstandigheden in Wi-Fi-netwerken aan. Elke stroom kan niet afzonderlijk worden gepland in de tussenliggende netwerkknooppunten. Door de beslissingen op het eindapparaat zelf te nemen, kunnen de tussenliggende knooppunten dus alleen het aantal stromen en hun toewijzing bijhouden. Door de geaggregeerde stroominformatie zoals capaciteit, airtime, enz. te communiceren, wordt de privacy van andere eindapparaten beschermd. De acties die door het eindapparaat worden ondernomen, zijn impliciet van aard en het tussenliggende knooppunt deelt niet expliciet de details over andere eindapparaten in het netwerk. Daarom kan het gebruik van rijkere netwerkinzichten en het impliciet uitvoeren van acties op het eindapparaat de netwerkprestaties aan-

zienlijk verbeteren, terwijl het zich aanpast aan een onvoorspelbaar veranderende draadloze omgeving.

De voorgestelde ontwerpen en discussies in dit proefschrift beloven de haalbaarheid van adaptieve transportlaagprotocollen in particuliere professionele draadloze netwerken, die tegemoetkomen aan de uiteenlopende eisen van diverse toepassingen, terwijl ze zich aanpassen aan veranderende netwerkomstandigheden. De talloze gegevens die via deze tests worden verzameld, kunnen worden gebruikt om adaptieve applicatielaagprotocollen van de volgende generatie te bestuderen en te implementeren.

Summary

Connectivity is the key tool in today's world. Especially during the COVID pandemic, connectivity has played a vital role in keeping critical communications running. Connectivity is ingrained in various sectors like industry, manufacturing, healthcare, finance, military, etc. through the wide adoption of private-professional wireless networks. This has tremendously increased the number of devices connected to the network. In addition to that, the diverse applications such as cloud computing, AR/VR, ubiquitous networking, human-to-machine interaction, etc., in these networks have assorted demands in terms of throughput and latency. Additionally, a few applications also have time-critical requirements, which need to be time-sensitive and deterministic. The impact of these events has resulted in a large number of heterogenous traffic flows in the network. The promise of high data throughput and lower latencies in the recent generations of Wi-Fi standards has increased its utilization, and it is among the widely adopted communication technologies in private-professional networks. Moreover, Wi-Fi is recently being utilized even for time-sensitive networking. The stringent demands of private networks and time-sensitive networking are now amplified by the challenges posed by wireless technologies.

Recent innovation in network management has adopted SDN to separate the control and data plane, enabling remote network reconfiguration. P4 programming is used for flexibility in the forwarding plane. SDN's expansion to the wireless domain has led to frameworks for data plane programmability in Wi-Fi and 5G. In addition to novel time-sensitive networking enablers, having such data plane programmability can help to achieve deterministic networking in mixed wired and wireless networks, paving the way for service differentiation and application prioritization. However, understanding the network's state and application needs is crucial to fully leveraging programmability. Despite having a wide range of applications, they are confined to choosing a limited number of quality-of-service options. The forwarding priorities offered by the existing network components are static in nature and do not offer sufficient granularity. Therefore, to enable the configurability and flexibility in the networks and to adapt and reconfigure the network to the application requirements, it is essential to involve the application layer in managing the network by acknowledging its requirements to the network layer. The APP-NET interface allows applications to specify their traffic and monitoring requirements to the network layer, which can be used to configure the underlying network. To understand the network state in real-time, it is essential to continuously monitor and verify the network performance. The limitations of

out-of-band methods for measurement and evaluation have led to an increased focus on INT. Originally limited to switched wired networks, the idea has now been applied to wireless systems as well. The INT framework for wireless networks has proven to be low-overhead, fine-grained, and reliable. INT can collect end-to-end and hop-by-hop node information along with real-time wireless link information.

Sufficient information about the application requirements and network condition can be gathered and utilized with data plane programmability, APP-NET interaction, and advanced telemetry like INT. However, the existing time-sensitive scheduling and slicing algorithms can only cater to the time-critical flows. When the number of flows in the network approaches hundreds or even thousands, it becomes difficult for intermediate network nodes to maintain a finely tuned, differentiated treatment for each flow while accommodating more functionality (such as traffic shaping, queueing, time sensitivity, etc.), eventually surpassing their capacity to support extra features.

As the computing power and memory of end devices are increasing, and some of them now have the capacity for analytics and intelligent decision-making using artificial intelligence, they can be utilized in supporting optimal network functioning. Given the aforementioned constraints on the fine-grained handling of flows and the growing capabilities of end devices, it is appropriate to ask how and to what degree end devices, specifically their transport layer protocols, can cooperate to jointly achieve optimal network functioning.

The transport layer of the OSI model provides logical host-to-host communication. Unlike UDP, TCP provides reliable, error-free, ordered, guaranteed data transfer service using congestion and flow control mechanisms. Several studies have focused on improving congestion control algorithms for better network performance. Traditional congestion control algorithms detect issues like packet loss, increased delay, or use hybrid approaches based on limited end-to-end information. To reduce latency and optimize network capacity, new feedback techniques are used. Google's QUIC protocol, for example, uses spin-bit packets to detect congestion. While some congestion control algorithms use the ECN bit in the IP header. However, ECN has limitations, including delayed notifications and incompatibility with switches. The new L4S protocol reinterprets ECN for low-latency performance but requires dedicated router queues and can conflict with traditional congestion control algorithms. Additionally, the existing congestion control algorithms are designed to exploit the high bandwidth of wired networks and maximize their throughput. In the case of wireless, there are other challenges like lossy medium, interference, changing channel conditions, etc. Lately, there has been increased interest in learning-based congestion control algorithms. But these algorithms still do not use any network feedback or use limited feedback through ECN. Moreover, these algorithms are trained offline, which contradicts the changing and unpredictability of wireless links.

This doctoral thesis addresses the above issues from the perspective of higher-layer network protocols by involving them in the network decision-making. With the recent innovations in wireless networks such as network telemetry, application-network interaction, data programmability of the intermediate network nodes, and

programmability of the kernel through eBPF, this thesis proposes next-generation adaptive congestion control algorithms for private-professional networks. Initially, the feasibility of adaptive transport layer protocols is investigated by finding the relation between real-time network parameters from INT and the congestion window size. eBPF is integrated with INT to synchronize the amount of transmitted TCP data with INT information. The results from this study indicated a direct influence of the behavior of congestion window size on the key parameters of intermediate network nodes, such as packet arrival rate, buffer queue capacity, and number of flows. Using the INT information, the existing CUBIC congestion control algorithm was modified to consider packet loss due to the wireless link conditions. The improved performance of the algorithm validated the potential of detailed insight into the network state.

Further exploiting the relationship between congestion control algorithms and network context, a network- and application-aware congestion control algorithm is designed. Utilizing the data programmability of the forwarding plane, the intermediate node is configured to process the application requests (facilitated by APP-NET and INT) and allocate capacity and priority to each traffic flow in the network. This allocation is further communicated to the congestion control algorithm of the end device, and the algorithm modifies the data transfer rate based on this feedback and the real-time network context. Hence the designed algorithm is aware of both application requirements and changing network states. In addition to congestion-free service, the algorithm considers the application needs and provides fine-grained service differentiation, thus taking over some of the burden of intermediate nodes. The algorithm also outperforms the existing CUBIC congestion control algorithm.

To address the issue of throughput degradation in Wi-Fi networks due to a fair channel access mechanism, and achieve airtime fairness, a reactive and adaptive congestion control algorithm is designed. The access point explicitly allocated the airtime to each end device solely based on the number of end devices in the network. Using this, the algorithm adapted its application data transfer rate. The reactive nature of the congestion control algorithm resulted in underutlization of the wireless network channel. To mitigate this a proactive congesiton control algorithm is designed.

This algorithm is a feedback-based PI control loop congestion control algorithm. The intermediate node is configured to keep track of airtime usage of each end device and communicate the aggregated airtime information. The designed congestion control algorithm utilizes this information to calculate the error in airtime usage, thus correcting the number of data packets to be shared during the next data transfer. Unlike the existing congestion control algorithms, the designed algorithm achieves airtime fairness and is robust to changing channel quality as well as other network parameters by adapting applications data transfer rate.

Though the designed algorithms are adaptable to the wireless network conditions, it is beneficial to be able to anticipate the future network. The PI control-loop-based congestion control algorithm is already proactive in nature, but to further extend the capabilities to application requirements and to explore the learning meth-

ods, a reinforcement learning-based approach was investigated. Since the wireless network parameters such as channel quality, other competing nodes, and number of flows, being independent of the actions taken by an end device, have a significant influence on the overall throughput and latency of the end device, the multi-context-aware RL-based congestion control is modeled as a contextual Markov Decision Process. Three different multi-context-aware RL-based congestion control algorithms are introduced that adapt the number of application data packets sent to the network based on application requirements and real-time network information. Two of these solutions were implemented and tested in the Wi-Fi network.

All the designed congestion control algorithms are implemented on commercially available wireless devices, and the validation against existing congestion control algorithms was conducted in the WiLab2 testbed of IDLab. The better performance of designed algorithms in terms of quality of service, airtime fairness, and adaptability to wireless conditions assures the importance of real-time network knowledge in higher-layer protocols. In addition to providing congestion-free services, the designed algorithms facilitate airtime fairness, application requirements, and adapt to changing network conditions in Wi-Fi networks. Each flow cannot be individually scheduled in the intermediate network nodes. Hence, by taking the decisions at the end device itself, the intermediate nodes can just keep track of the number of flows and their allocation. By communicating the aggregated flow information such as capacity, airtime, etc., the privacy of other end devices is protected. The actions taken by the end device are implicit in nature, and the intermediate node does not explicitly share the details about other end devices in the network. Therefore, using richer network insights and taking actions implicitly at the end device can significantly improve the network performance while adapting to an unpredictably changing wireless environment.

The proposed designs and discussions in this thesis promise the feasibility of adaptive transport layer protocols in private-professional wireless networks, catering to the varied demands of diverse application whilst adapting to changing network conditions. The myriad data collected through these tests can be utilized to study and implement next-generation adaptive application layer protocols.

1 Introduction

"You have the right to work but never to the fruit of work."

– Bhagavad Geeta

This chapter focuses on the context behind the research done in this thesis. It starts with a brief introduction to the importance of connectivity and connectivity solutions for private-professional networks. Further, a summary of the importance of end-to-end networking and the impact of congestion control algorithms on network performance and management is discussed. This is followed by a description of recent innovations and concepts in wireless networking, such as data programmability, in-band network telemetry, and application-network interaction. Next to this, a list of targeted challenges is discussed with the research contribution of this thesis to solving each of them. Finally, the outline of this thesis and the list of papers authored during the research period are provided.

1.1 Context

1.1.1 The importance of connectivity

Connectivity is a general term used to describe being connected to an entity. In the networking field, it means the possibility of one device being connected to the network. Technologies like Wireless Fidelity (Wi-Fi), mobile networks, and broadband that enable exchange of data between the end devices facilitate connectivity. Connectivity provides instant communication across the world through calls,

video conferencing, and messaging and connects the entire world through social media. It provides access to information and helps in trading and e-commerce applications. Most importantly, one of the crucial features that categorizes things as smart is the ability to be connected and to share information. Connectivity is essential for the Internet of Things (IoT), which in turn supports diverse use cases such as smart cities, smart homes, smart factories, Industry 4.0, smart healthcare, etc. Therefore, connectivity is a vital tool in today's world. Such importance was proved during the COVID pandemic, when the communication networks kept the critical communications running.

The progress in connectivity and communication networks has resulted in tremendous increases in the number of connected end devices, resulting in an increase in the number of traffic flows in the network. It is predicted that the number of devices connected to the network will double between 2023 and 2029, reaching 50 billion devices by 2029[1]. More and more diverse connected devices boosts the number of flows as well as their diversity in terms of requirements. This number is heavily influenced by the wide adoption of connectivity in sectors like industry, manufacturing, ubiquitous computing, human-to-machine communication, energy, etc. These sectors demand enhanced capabilities of connectivity solutions in terms of coverage, performance, reliability, latency, throughput, security, and most importantly, customization. Coverage is essential to connect to the remote devices that are quite common in sectors like manufacturing and energy. These networks also demand better performance in terms of latency and throughput from the network. One of the essential features is reliability for consistent and guaranteed performance in sectors like healthcare and human-to-machine communication. These networks are required to be secure, supporting restricted access for specific users to critical communication infrastructure. Most essentially, communication networks need to be innovation-friendly and customizable to cater to the diverse needs of these sectors. The ever-increasing demand for connectivity and evolving technologies in edge computing, Artificial Intelligence (AI), and IoT have resulted in the wider incorporation of private-professional wireless networks for dedicated usage in several sectors.

1.1.2 Private-professional networks: A connectivity perspective

A private-professional network is a custom network infrastructure built for an individual user or user group. These networks are made to give complete control over network data and to provide dependable, secure wireless communication. The private-professional networks were initially deployed to deliver radio coverage in distant geographical locations. Land Mobile Radio (LMR) systems and 2G

[1]Statistics on number of connected devices worldwide in 2015 and 2029, by device: https://www.statista.com/statistics/512650/worldwide-connected-devices-amount/

technologies were frequently used in these antiquated systems. Further, private-professional wireless network pioneers were usually found in industries like utilities, transportation, and public safety. These early versions did, however, have several shortcomings. These networks were unsuitable for emerging applications because they were inflexible, lacked scalability, and were frequently difficult to scale. These networks had limited data capacities and were expensive to maintain and upgrade, which presented further difficulties.

A significant paradigm shift in private wireless networks was brought about with the introduction of Long-Term Evolution (LTE) and 5G technology. Better features like higher data rates, better scalability, and superior Quality of Service (QoS) were brought forth by the integration of LTE. It was a critical turning point in the development of private networks by enabling them to support a wider variety of data-intensive applications. Next to cellular (4G/5G), Wi-Fi is a key technology for the deployment of private-professional networks, in particular for indoor use cases. The compound annual growth rate of private wireless networks is expected to increase by 62% between 2024 and 2030[2], showing an increase in adoption of private-professional networks over this period.

The driving component of private-professional networks is their support for diverse applications and room for customization and innovation. Video conferencing, instant messaging, financing tools, and email are some of the applications in the communication and business sector verticals. Heating, ventilation, air conditioning, lighting, security, control devices, predictive maintenance, human-machine interaction, and production processes are some of the applications in the industrial automation sector vertical. Augmented reality/ Virtual reality (AR/VR), fleet management, inventory tracking, and warehouse automation are some of the logistics applications. Edge computing, predictive analytics, and real-time automation are some other applications. All these diverse applications should be supported by the private-professional networks, considering their diverse characteristics and network performance requirements.

Data-intensive applications like video conferencing, high-definition content for media, cloud-based software, and edge computing demand high bandwidth and low latency. Applications in industries, healthcare, and logistics require real-time communication and monitoring, relying on critical communication infrastructure. Certain applications, like communication between machines and robots in industrial automation, real-time data communication in flights, and secure communication in mission-critical systems, require high reliability and bounded latency, thus time sensitivity and determinism. Based on these requirements, it is evident that, in addition to diverse requirements, the application demands are stringent in nature.

Hence, in private wireless networks, end-to-end dependable communication is

[2]Private networks market forecast 2024 – 2030: https://stlpartners.com/research/private-networks-market-forecast-2024-2030/

essential to guaranteeing timely data delivery without loss or corruption. This is crucial for critical applications like real-time communication tools and IoT devices that cannot tolerate losses or late delivery of packets. Networks supporting such requirements are called time-sensitive networks or deterministic networks.

1.1.3 Time-sensitive networking

In private-professional networks, an emerging requirement is guaranteed and predictable network performance for novel applications like robot communication, industrial IoT, industrial automation, AR/VR, aerospace, etc. In addition to application-dependent throughput, they also demand guaranteed ultra-low latency and ultra-reliability. Such applications are called time-sensitive applications. The network's ability to support such applications is known as Time-Sensitive Networking (TSN) [1].

The IEEE 802.1 TSN working group initially developed TSN for wired Ethernet networks. Many of its standards have now been adopted by manufacturing, automotive, and industrial automation. The most significant TSN characteristics and difficulties are summed up in a thorough assessment [2] as follows: precise time synchronization, traffic shaping and scheduling, frame replication, and network management. In the following, we describe in more detail each of the main TSN enablers.

- **Precise time synchronization:** TSN relies on highly accurate time synchronization among devices in the network, often done via IEEE 802.1AS standard [3]. Since every device uses the same clock, they can all precisely coordinate communication down to the nanoseconds. Applications that need deterministic behavior, like autonomous transportation or industrial automation, utilize such synchronization to support traffic organization in the network.

- **Frame replication:** To improve network reliability, TSN makes use of frame replication and deletion. In the network, data frames are copied and transferred across several distinct paths. The other copy guarantees data transmission in the event that one path malfunctions or is delayed, lowering the possibility of packet loss and enhancing fault tolerance.

- **Network management:** To configure scheduling strategies, traffic prioritization, and time synchronization, TSN networks need powerful management and configuration tools. The management and control framework is standardized under IEEE 802.1Qcc [4]. It specifies three different network management topologies: fully centralized, fully distributed, and centralized network management/distributed user control. Such management mechanisms ensure network performance in accordance with the intended real-time

requirement of applications while dynamically adapting to the network and traffic flow changes.

- **Traffic shaping and scheduling:** Traffic shaping and scheduling are essential methods in TSN that are used to control data flow and guarantee that time-sensitive applications (such as audio-video systems, autonomous vehicles, or industrial automation) adhere to stringent latency, jitter, and bandwidth requirements. There are several scheduling and traffic shaper mechanisms that are already standardized for wired networks, such as: time aware shaper, credit-based shaper, cyclic queuing and forwarding.

1.1.4 Scheduling enablers in TSN

This subsection gives an overview on generally used traffic shaping and scheduling mechanisms in TSN.

The exact temporal control of packet transmission to satisfy applications' demands for real-time performance is referred to as scheduling. The scheduler makes sure that every data stream complies with the timing specifications of the network by transmitting it at the appropriate time and in the appropriate order. Some of the commonly used scheduling techniques are:

- **Credit-Based Shaper (CBS), or IEEE 802.1Qav**, uses traffic credits to regulate the stream transmission rate [5]. The credit value is increased when the channel is busy and the buffer already has packets, while it decreases the credit while the node is using the channel to dequeue packets. By doing this, traffic flow is balanced, data transfer is smoothed, and buffer overflows are prevented. Applications with stringent bandwidth needs, like professional audio-video streaming, can use it.

- **Time-Aware Shaper (TAS), or IEEE 802.1Qbv**, divides the time into cyclic intervals where distinct time slots are assigned to various traffic classes [6]. High-priority industrial control messages and other critical traffic are given dedicated time periods during which no other traffic is allowed to interfere. Time-sensitive traffic is guaranteed to always be transmitted on schedule and unaffected by lower-priority traffic due to the deterministic behavior of the TAS.

- **Asynchronous Traffic Shaping (ATS), or IEEE 802.1Qcr**, with its more sophisticated approach, ensures low latency and consistent forwarding throughout the network, even for best-effort traffic, by managing packet forwarding based on arrival times [7]. Without strictly maintaining synchronization, it adapts dynamically to the state of the network to optimize for both high-priority and lower-priority traffic.

- **Per-Stream Filtering and Policing (PSFP), IEEE 802.1Qci,** makes sure that every stream abides by the bandwidth and timing restrictions on the network, by enforcing rules on a per-stream basis [8]. Traffic is filtered and policed, with packets not meeting schedule or rate constraints being dropped. This is especially helpful in avoiding scheduled transmissions being disrupted by traffic bursts.

- **Frame Preemption, IEEE 802.1Qbu/802.3br,** enables low-priority frames to be stopped in the middle of transmission, when a higher-priority frame needs to be sent [9]. The low-priority frame continues after the essential frame is sent. This guarantees that even when there are big best-effort packets in the transmission queue, time-sensitive traffic won't be delayed.

Summary on TSN traffic shaping and scheduling: To provide a comprehensive solution for managing diverse types of traffic (critical and non-critical) on a single network, TSN frequently integrates both traffic shaping and scheduling. To ensure real-time assurances, critical traffic is prioritized and subject to stringent scheduling (TAS) and shaping (CBS). In order to manage best-effort traffic and ensure that it does not interfere with time-sensitive streams while still making use of network capacity, additional shaping techniques like CBS and frame preemption are used. Deterministic behavior with guaranteed on-time delivery for important traffic and low, predictable latency are benefits of TSN traffic shaping and scheduling. The network can handle best-effort and time-sensitive traffic without overprovisioning by reshaping the traffic. TSN mechanisms are adaptable enough to meet a variety of application needs in a variety of sectors. Difficulties include the intricacy of setting up and maintaining TSN schedules, which call for exact planning and network-wide cooperation. It is challenging to make sure that TSN standards are compatible with the current network architecture, particularly in situations with mixed wired-wireless traffic.

The expansion of TSN to wireless is the next logical step in its progress, since many new applications require the flexibility that wireless offers. 3GPP 5G and IEEE 802.11 (Wi-Fi) are being evaluated as two main candidate technologies to enable Wireless-TSN (W-TSN) since both standards can provide high data rates, which is a first requirement to lower the packet transmission latency itself. However, a number of significant obstacles still need to be overcome in order to introduce TSN capabilities: obtaining ultra-low bounded latency; effectively scheduling transmissions over a shared medium (for time-sensitive and non-time-sensitive flows with the possibility of cross-node transmission preemption); achieving zero-loss and deterministic inter-channel handovers; and managing the network and spectrum utilization [10]. The scheduling problem seeks to determine appropriate paths and traffic shaper configurations for wired TSNs. It continues to be particularly difficult, with approximative methods like heuristics, genetic algorithms, machine learning,

and exact methods like integer/mixed linear programming and constraint programming approaches being researched [11]. The scheduling in W-TSN networks where transmission over a shared medium results in overlapping of the data transmission is indicated in Figure 1.1. Since the wireless channel might change over time, the data rate utilized by nodes will change as well. This will impact the time on air of the packets, requiring longer time slots in case a TAS scheduling mechanism is used. In such a case, the end-to-end wired-wireless scheduling needs to be updated in order to avoid a long residual time of packets at certain hops due to time slot misalignment. This will require frequent scheduling updates for the dynamically wireless channels, posing another challenge for implemented TSN in wireless.

1.1.5 Wi-Fi for TSN in private-professional networks

The IEEE 802.11 standard, created in 1997, served as the foundation for the late 1990s emergence of Wi-Fi, short for Wireless Fidelity. The first version was limited to 2 Mbps in speed. With the establishment of the Wi-Fi Alliance in 1999, an industry association that certifies products and promotes wireless Local Area Network (LAN) technology, Wi-Fi gained widespread acceptance. With download speeds of up to 11 Mbps, 802.11b was the first widely used Wi-Fi standard, introduced in 1999. Newer standards like 802.11g, which offered rates of up to 54 Mbps in 2003, marked a rapid evolution in Wi-Fi technology. Later iterations, like 802.11n (2009, rate upto 600 Mbps) and 802.11ac (2013, rate upto 1.76 Gbps), greatly enhanced dependability, data rates, and range. Wi-Fi 6 (802.11ax) was introduced in 2019 and has since improved in latency, capacity, and operation in crowded environments, making it appropriate for high-density situations and IoT, offering rates upto 2.4 Gbps. Wi-Fi 7 is expected to provide multi-gigabit throughput upto 46 Gbps and substantially increase speeds. The Wi-Fi networks are expected to have an increase in compound annual growth rate of 20% between 2024 and 2032[3] only in North America. Even with the wide spread of such networks in well-developed countries, a further increase in adoption is expected in the next few years as well.

Such an increase in utilization of Wi-Fi in the upcoming years comes as a result of its capabilities to be used in dedicated private-professional networks. Wi-Fi networks are ideal for locations like warehouses, residences, and enterprises because they are easy to set up using readily available hardware such as routers and Access Points (APs). Compared to cellular networks, they are cheaper to deploy and maintain as they do not require spectrum licenses or extensive infrastructure. The Wi-Fi devices offer broad compatibility with other end devices. Wi-Fi networks can achieve multi-gigabit rates with Wi-Fi 6 and Wi-Fi 7, which makes them

[3]Wi-Fi as a Service Market Size, Share & Industry analysis: https://www.fortunebusinessinsights.com/wifi-as-a-service-market-104859

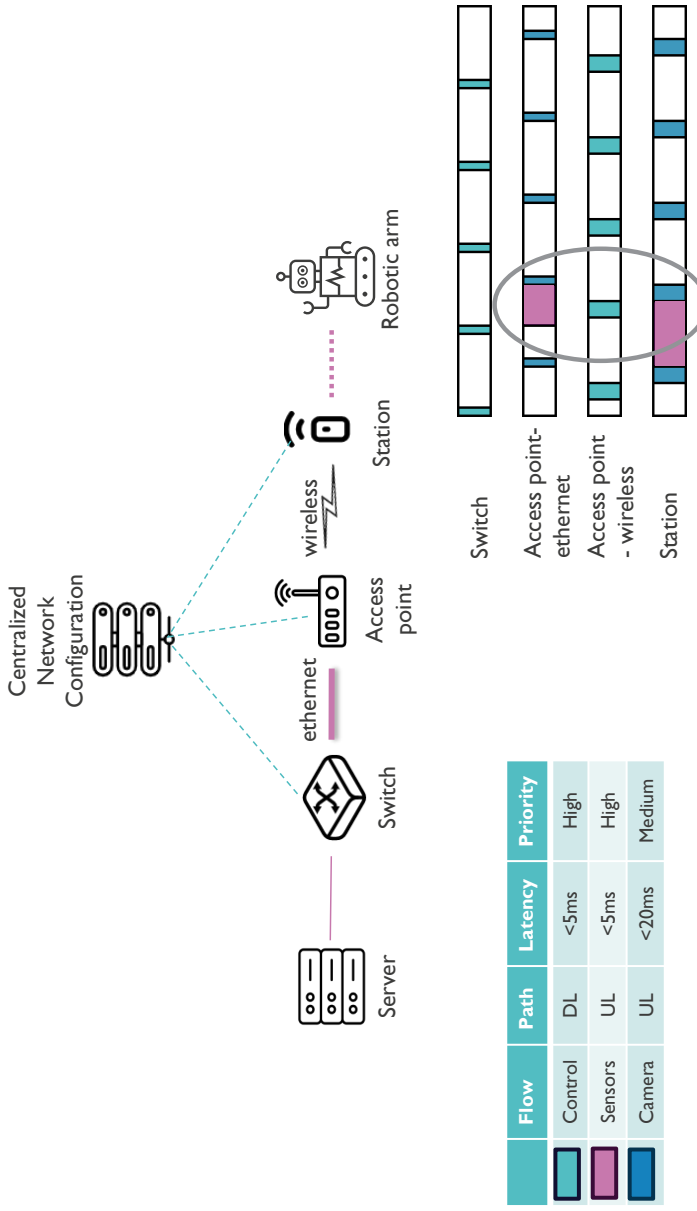


Figure 1.1: Scheduling in TSN, indicating the overlap in data transmission in shared timeslot

appropriate for data-intensive applications like video conferencing. Since private Wi-Fi networks allow sectors to manage and control their network traffic with full ownership, it allows room for customization. The above-outlined and many other features of Wi-Fi have resulted in vast adoption of this technology for private-professional networks.

Wi-Fi for TSN: As mentioned earlier, the promises of future Wi-Fi technologies have made it a suitable candidate for TSN. The upcoming (IEEE 802.11bn or Wi-Fi 8) and almost finished (IEEE 802.11be or Wi-Fi 7) modifications, which include significant features that further enhance latency and reliability, partially address W-TSN-related issues [12]. These new features include beamforming, Restricted-Target Wake Time (R-TWT), Multi-Link Operation (MLO), joint transmissions, Multi-AP coordination (MAP-co) for spatial reuse [13], OFDMA [14], and other enhancements for deterministic communication, such as resource reservation, (cross-node) preemption [15], and enhanced Enhanced Distributed Channel Access (EDCA). TSN might be one of the use cases for 802.11bn, while its specifics are still up for debate [16].

The research community is already using software-defined radio platforms to enable W-TSN by integrating time-sensitive networking concepts into Wi-Fi. Openwifi is a full-stack Wi-Fi design and implementation platform compatible with Linux that can operate on multiple FPGA platforms; it is being used for research in W-TSN [17]. The researchers have been able to integrate TSN features like hardware-based timestamping [18], microsecond-accurate synchronization, an IEEE 802.1Qbv-like gating mechanism, impactless bootstrapping [19], and mobility [20] because of Openwifi's open and customizable design, which extends from the driver to the digital baseband. Though very promising, the feature set still has to address missing W-TSN capabilities and include carefully chosen Wi-Fi aspects necessary for W-TSN. In addition, contention problems that hinder real determinism need to be removed and TSN scheduling algorithms need to be redesigned to manage the unique dynamics of the shared wireless media. The latter problem results from using unlicensed airwaves for operations.

The scheduling algorithms can only cater to the needs of time-critical flows, ignoring the priorities of other flows. Achieving fine-grained scheduling for each and every flow inserts additional burden on the network nodes specially with drastic increase in the number of flows.

1.2 Network from a transport layer perspective

From the internet to private-professional systems, end-to-end network protocols are essential for providing dependable, secure, and effective communication across a variety of networks. These protocols control the flow of data between two or more endpoints, which can be any number of devices, systems, or applications. They

make sure that data is securely transferred, accurately delivered, and appropriately maintained at every step of the way.

The end-to-end protocols can be used to ensure guaranteed delivery. In order to maintain data integrity, they also identify errors during transmission and take correction action to fix them. This is crucial in settings where precise data is essential, such as financial transactions or the interchange of medical data. End-to-end protocols like Internet Protocol Security (IPsec) and Transport Layer Security (TLS) guarantee the confidentiality, integrity, and tamper-proofness of data transferred over a network. Applications such as banking, healthcare, and military communications, where security is crucial, require this. By verifying the parties to a communication and limiting the exchange of data to reliable sources, these protocols lower the possibility of fraud or illegal access.

Protocols like User Datagram Protocol (UDP), used in real-time applications like gaming, voice calls, and live streaming, are designed to minimize delays in data transmission by sacrificing reliability for speed. This is important in use cases like AR/VR, live video, and interactive applications. End-to-end protocols are made to be flexible enough to adjust to various network scenarios and settings, guaranteeing reliable performance at diverse latencies, bandwidths, and network configurations (wired, wireless, mobile). Protocols like Transmission Control Protocol (TCP) and Quick UDP Internet Connections (QUIC) support many applications, from online browsing to video streaming, by efficiently handling several simultaneous connections and varied types of data flows.

Devices from different manufacturers running different operating systems can communicate easily over a network thanks to protocols like Internet Protocol (IP). In the context of the global internet and expansive private networks, such as those in finance and healthcare, this is crucial. End-to-end protocols control the data transmission, reception, and display, which has a direct effect on the user experience. For instance, Real-time Transport Protocol (RTP) enables lag-free video streaming and communications, while HTTP/HTTPS (based on TCP) ensures that web pages load correctly and securely.

End-to-end protocols provide end-to-end communication services with, among other functionalities, reliability support and congestion avoidance. In private-professional networks with mixed traffic, they can provide such end-to-end communication, but nothing more. When services are time-sensitive, one can revert to TSN, but here there are limits in terms of scheduling per flow. So still a role for these protocols to play. And lastly, when wireless is introduced, the specifics of wireless also pose challenges that can affect the functioning of the transport protocols. So it is good to take a look at current transport protocol design before then diving into the specific challenges.

1.2.1 Transport layer protocols

The transport layer lies between the application and network layer of the network Open Systems Interconnection (OSI) model. It is responsible for logical end-to-end communication between the two end devices. The transport layer provides different services, such as reliable data transfer, flow control, Congestion Control (CC), error detection, etc., to the higher application layer. Two widely used transport layer protocols are TCP and UDP.

TCP is connection-oriented and provides reliable, error-free, ordered data transfer services. TCP can provide these services utilizing its CC and flow control mechanisms. CC decides the number of application packets sent at each instance while avoiding congestion anywhere in the network; flow control prevents the buffer overflow at the receiver. TCP is preferred for the applications that require reliable communication, such as HTTP, file transfer, streaming media, and some industrial applications such as supervisory control and data acquisition, programmable logic controllers, predictive maintenance systems, etc. Unlike TCP, UDP is connection-less and does not provide ordered, error-free, guaranteed delivery. UDP is lightweight and is preferred for applications that can tolerate packet loss.

QUIC is a transport protocol designed by Google that is intended to offer similar services as TCP; however, the connectivity is quicker and more effective than TCP [21]. Built on top of UDP, QUIC addresses some of TCP's shortcomings, especially with regard to latency and data rate in contemporary online applications, while incorporating capabilities commonly present in TCP, such as dependability and CC. Since QUIC is based on UDP, it has problems with networks that throttle or prohibit UDP traffic. This could affect QUIC's availability and performance in comparison to TCP, which is supported by all networks. Although QUIC uses a similar CC mechanism to TCP, it is not as developed. Network infrastructure has long supported TCP, and it is well established. On the other hand, because QUIC's encrypted headers prohibit network devices from examining and optimizing traffic (as they frequently do with TCP), it can be more difficult to diagnose. The inability to see QUIC traffic can lead to problems with proxies, firewalls, and other middleboxes. Since QUIC's encryption utilizes each packet as a mini-encrypted frame, it requires more CPU and memory than TCP.

Due to these drawbacks, TCP is still a preferable transport layer protocol. Assuming the end device has infinite buffer space, the principal algorithm that provides the services of TCP is the CC algorithm. The CC algorithm decides the number of application packets that are sent to the network at each instance while avoiding network congestion at each network node in between.

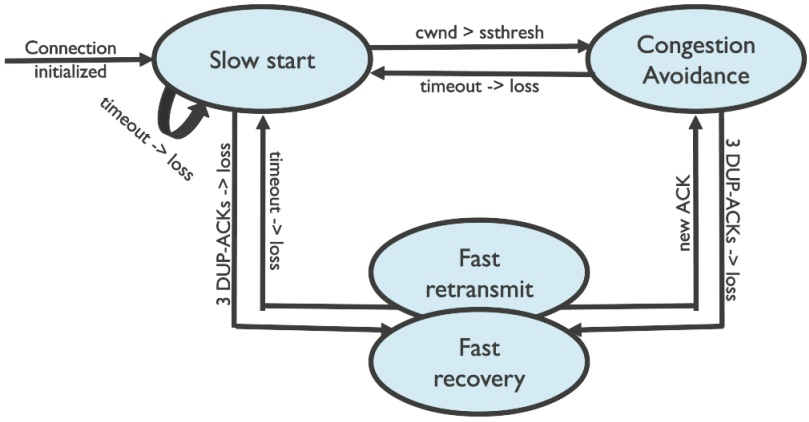


Figure 1.2: Different phases of CC mechanisms

1.2.2 CC algorithm

There are several CC algorithms, of which TCP Reno and TCP CUBIC are the most widely used. Before giving a brief description of each of these algorithms, a short introduction to different phases of CC algorithms is given here.

TCP flows include a series of data packets sent from a sender to a receiver, along with a corresponding series of acknowledgment (ACK) packets flowing in the reverse direction. At any given time, a sender may send a certain number of packets (known as the congestion window size, or $cwnd$, a small integer multiple of the Maximum Segment Size (MSS) allowed on that connection) before an ACK. Thus, the size of the $cwnd$ controls the rate of data sent on a flow. Using TCP CC procedures, a source increases or decreases a flow's $cwnd$ based on the network events. The triggering factor for CC algorithms is the packet loss event, which can occur due to a timeout or three Duplicate Acknowledgements (DUP-ACKs). Every CC mechanism has four phases: slow start, congestion avoidance, fast retransmit, and fast recovery. Each algorithm enters the slow start phase after a connection is initialized or after a timeout. In this phase, for every packet ACKed, the $cwnd$ increases by one MSS; hence, the rate of increase is very rapid, and $cwnd$ doubles for every Round Trip Time (RTT). When the $cwnd$ is more than the slow start threshold ($ssthresh$), the algorithm enters the congestion avoidance phase. Every time a packet loss event is detected by 3 DUP-ACKs, the TCP performs fast retransmit, where it retransmits the missing packet and enters the fast recovery phase. Figure 1.2 demonstrates how the algorithm switches from one phase to another.

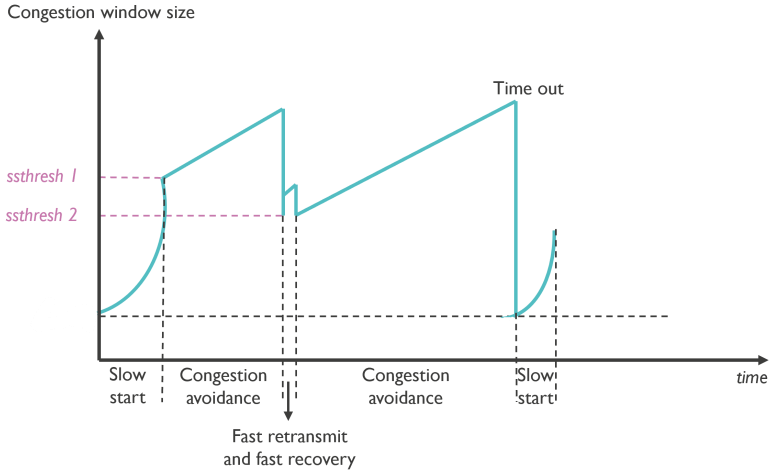


Figure 1.3: Different phases of TCP Reno

CC mechanisms are different by their behavior or actions taken during each congestion phase. In the following, we explain the most widely used CC algorithms in today's networks: TCP Reno, TCP CUBIC and TCP BBR.

TCP Reno was initially introduced in the 1990 Berkeley Software Distribution (BSD) release, after which several versions have been introduced. Currently, New Reno [22] is the recent modification that is available in the Linux kernel. Like every algorithm, Reno starts with the slow start phase, the sender starts with a small $cwnd$ and increases it exponentially with each ACK received, doubling $cwnd$ each RTT until the *ssthresh* is reached. In congestion avoidance phase, once $cwnd$ exceeds *ssthresh*, the sender increases $cwnd$ linearly (one MSS per RTT) to avoid overwhelming the network. If three duplicate ACKs are received, indicating a packet loss, the sender retransmits the lost packet without waiting for a timeout, which is a fast retransmit phase. In fast recovery phase, after retransmitting, Reno sets *ssthresh* to half of $cwnd$, and $cwnd$ is reduced to *ssthresh*. It then continues in congestion avoidance mode. In contrast, NewReno stays in fast recovery until all lost packets are acknowledged, allowing for more efficient recovery when multiple packets are lost, thus improving retransmission during the fast recovery phase. It introduces a specific algorithm to use partial ACKs and uses the TCP Selective Acknowledgement (SACK) option. Different phases of TCP Reno are indicated in the Figure 1.3.

TCP CUBIC was first implemented in 2006, and since then it has been used as a default algorithm in Linux machines [23]. The CUBIC algorithm behaves like Reno in slow start, fast retransmit, and fast recovery phases. The CUBIC algorithm is real-time dependent and modifies the $cwnd$ based on the elapsed time from the last

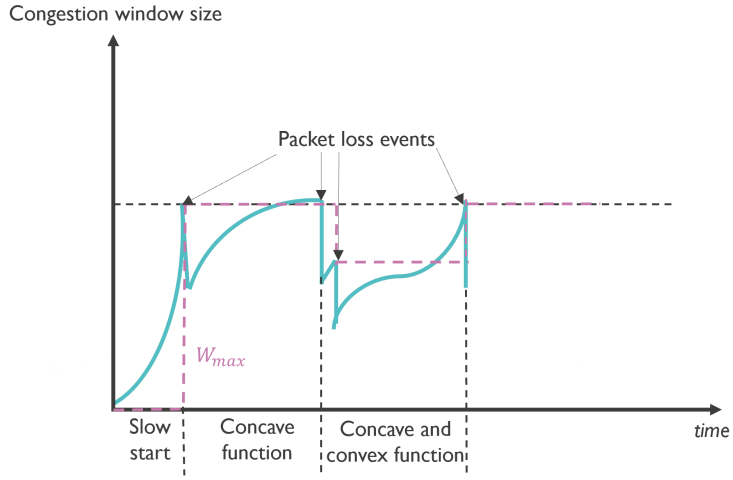


Figure 1.4: Different phases of TCP CUBIC [25]

congestion event. During the loss event, the $cwnd$ value is saved as W_max. After the algorithm enters congestion avoidance, the $cwnd$ is increased using the concave profile of the CUBIC function. In order to maintain the concave window rise until $cwnd$ becomes W_max, the CUBIC function is configured to plateau at W_max. After that, the convex window increase starts and the CUBIC function becomes a convex profile. The method stability is enhanced while retaining high network utilization because of the window adjustment (concave and then convex) [24]. These phases of CUBIC CC algorithm are shown in Figure 1.4.

TCP Bottleneck Bandwidth and Round-trip Propagation Time (BBR) is a contemporary CC technique that actively estimates the network's bottleneck bandwidth and RTT in order to optimize throughput and reduce latency [26]. BBR continuously assesses the network's capacity and modifies its sending rate to correspond with the available bandwidth, in contrast to conventional algorithms like TCP Reno, which mainly use packet loss as a congestion signal. Using a feedback loop, BBR determines the RTT and the expected bandwidth (based on the maximum rate at which packets may be transmitted without experiencing severe queuing). Then, in order to minimize queuing delays and prevent congestion, it modifies its $cwnd$ to maintain the ideal rate. By doing so, BBR aims to achieve higher performance, particularly in networks with high bandwidth-delay products, and in environments where packet loss is rare or delayed ACKs are frequent, such as in high-speed or long-distance networks.

To support congestion-free communication networks, there exist other mechanisms in the network layer as well. Explicit Congestion Notification (ECN) is a network feature that lets routers alert end devices about congestion without causing

packet loss. [27]. To manage network congestion more effectively, it is used in conjunction with the TCP protocol. Upon identifying congestion (such as a full queue), a router designates a unique ECN bit in the packet's IP header rather than discarding the packet. The receiver detects the flagged packet and alerts the sender about the congestion. To improve efficiency, the sender lowers its transmission rate in a manner akin to how it would respond to a packet loss, but without having to retransmit lost data. Using ECN, the end device experiences reduced packet loss, faster congestion response, and improved TCP performance by reducing traffic rates. Incorrect ECN bit handling by routers or other devices can result in less-than-ideal performance or even unintentional packet failures due to misconfigured ECN settings. In lightly populated networks, ECN's congestion signaling imposes needless complexity without meaningful benefits, as packet loss is already minimal. Moreover, ECN doesn't provide detailed insights into the network.

In order to reduce network congestion, the new RFC [28] for the CC algorithm reinterprets the ECN bits in the IP header and supports Low Latency, Low Loss, and Scalable throughput (L4S). But for L4S to function properly, routers must have separate queues for L4S-enabled traffic flows and other traffic flows. If traffic flows governed by standard CC algorithms and L4S mechanisms are mixed in the same queue, L4S may not function as intended.

The transport layer protocol is unaware of the underlying network conditions, and only has the perspective of end-to-end device connection. In case of TCP, the decisions taken by the algorithms of the protocol are either based on the partial information obtained from the ACK packets or partial notification from ECN.

1.2.3 Summary

In today's world, connectivity has become fundamental. Especially highlighted during the COVID pandemic, where it enabled essential communications across sectors like industry, manufacturing, healthcare, finance, and the military. The adoption of private-professional wireless networks has dramatically increased the number of devices connected to networks. Alongside this, diverse applications like cloud computing, AR/VR, ubiquitous networking, and human-machine interaction bring varied demands in terms of throughput and latency. These factors have led to a surge in heterogeneous traffic flows within networks. Wi-Fi, with its promise of high data throughput and low latency, has become one of the most widely adopted technologies for private-professional networks. However, the growing demands of these networks present challenges for wireless technologies. At the same time, end devices have become more powerful, with capabilities for analytics and intelligent decision-making through artificial intelligence. In light of these advancements, it is imperative to think about how end devices and transport layer protocols might cooperate to maximize network performance.

1.3 Recent innovations in wireless networking

Before addressing the challenges of private-professional wireless networks, it is important to look at recent innovations in wireless technologies and networking. These innovations are represented in Figure 1.5

1.3.1 Data programmability of forwarding plane

Greater control and efficiency in network traffic management are made possible by the advancement of data programming in the forwarding plane of wired and wireless networks, which has moved from static, hardware-defined methods to more flexible, software-driven systems. Earlier networks managed traffic flows using protocols (like IP) and hardware-based forwarding using fixed-function devices like switches and routers. To cater to the demands of diverse applications, recent technologies of wired networks have transitioned to Software Defined Networking (SDN) [29]. SDN splits the control and data plane, which allows remote reconfiguration of network devices. The complete programming of the forwarding plane is also enabled by the P4 programming language, which achieves flexibility in wireless networks [30]. This has also encouraged research in IEEE 802.11 (Wi-Fi) management [31] and 5G [32]. The flexibility of data plane programmability enables several possibilities, such as service differentiation, dynamic traffic management, application prioritization, and determinism. To leverage this, it is necessary to understand the application requirements and real-time network state.

1.3.2 In-band network telemetry

Continuous network performance monitoring and verification is one of the key features of future private-professional networks. Active and passive monitoring methods are the often used monitoring techniques where the probe packets [33] and network device polling [34, 35] are used respectively for collecting network information. Cisco Net-Flow [36], IP Flow Information Export (IPFIX) [37], and Simple Network Management Protocol (SNMP) [30] are few examples of protocols that use active and passive monitoring techniques. These monitoring methods introduce additional traffic in the network or collect information at fewer network nodes and are not fast enough to detect network state in real-time, as required by industrial applications. Due to the drawbacks of external measurement techniques, the focus has now shifted towards In-band Network Telemetry (INT).

Though INT was initially introduced for wired networks using P4 [38], lately it has gained wide attention for monitoring SDN applications. Several researchers have implemented and tested INT for SDN networks [39], [40], [41]. The first survey paper on INT [42] gives insights into the history, research areas, and application results of INT. It discusses several research works of INT for both wired and

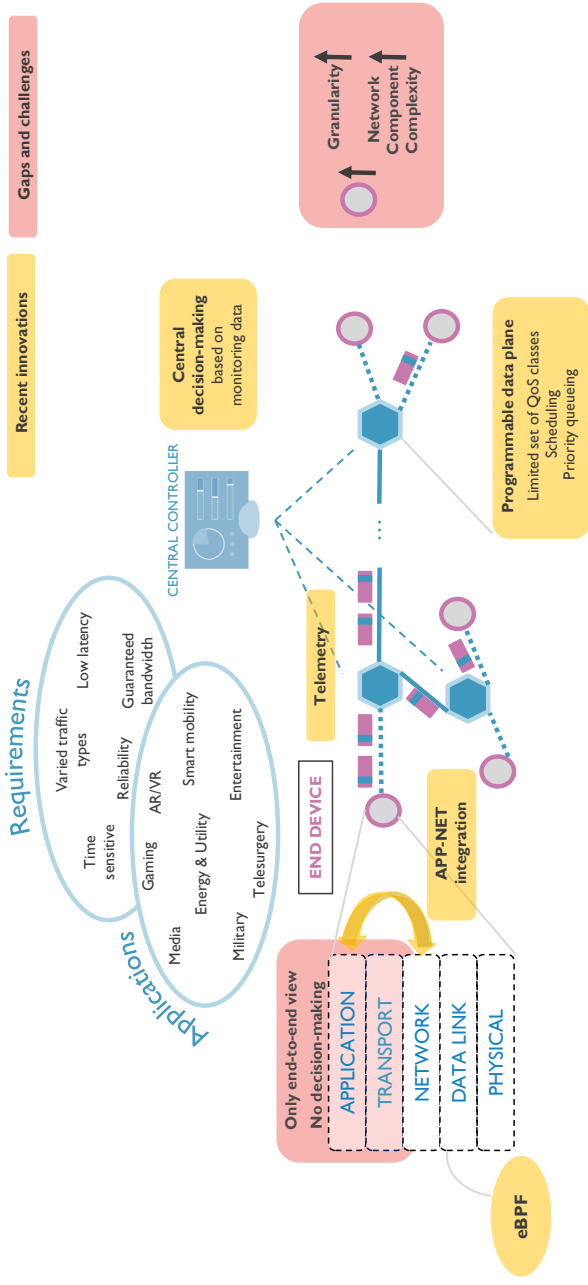


Figure 1.5: Research gaps and innovations

wireless domains and its latest applications. Using Alternate Marking-Performance Measurement (AM-PM), a hybrid telemetry for end-to-end network packet loss and delay measurement, with advantages of flexible deployment and high statistical accuracy, authors of [43] have designed a monitoring technique for end-to-end and hop-by-hop reliability and delay performance measurement of industrial wireless sensor networks. INT was extended from wired to the wireless network for industrial wireless sensor networks in [44]. In [45], authors have demonstrated INT monitoring techniques that gather fine-grained statistics via INT-enabled nodes and periodic statistics analysis against current application QoS requirements for an SDN-based framework. An INT-enabled node architecture and novel INT options for wireless networks is proposed in [46].

Along with designing an INT-enabled node architecture for wireless scenarios, authors of [47] have tested and validated the accuracy of the monitoring technique in terms of monitoring, overhead, and network (re)configuration. The INT technique designed in [47] had six times lower overhead than the existing active monitoring techniques. 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). The INT information is encapsulated in the IPv6 extension header and is received as feedback at the source node. Hence it is a low-overhead, continuous network monitoring technique that provides real-time network context to reconfigure the data transfer rate.

1.3.3 Application-Network Interaction

In the current OSI model, application and transport layers only have an end-to-end view of the communication and are uninformed of the underlying network. Previous efforts have aimed to make networks more application-aware [48–50]. The socket interface was expanded in [48] to allow applications to share their communication demands with the network stack, using ”socket intents.” Similar enhancements were made in [49] and [50], introducing a new API for sharing application demands with the transport layer and receiving transport performance feedback.

In addition to application requirements exposure functions, network monitoring functions need to be covered as well to support network-aware applications. In the Application-Network (APP-NET) interaction framework designed in [51], 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. 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 its identification, traffic and moni-

toring needs to the network, 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 which mainly consists of three parts, application identifiers (device ID, application ID, and node ID) and application properties (traffic types and network paths). Under traffic types, the application indicates the properties of the traffic generated (payload size, periodicity, priority). As designed in [51], the APP-REQ data is encapsulated in the IPv6 extension header. The APP-REQ is first encoded using Concise Binary Object Representation (CBOR) before encapsulation, to reduce the overhead. This information is sent as INT from the source to the destination during the connection initiation i.e., in the SYN packet. The framework also provides the possibility to further extend the interface to the transport layer through a transport adapter, resulting in application-transport-network layer interaction.

1.3.4 Extended Berkeley Packet Filters

To exploit these innovations, it is required to have the flexibility of programming protocol stacks. This is enabled by the recent innovation in the Linux kernel, extended Berkeley Packet Filter (eBPF) [52]. eBPF is a recent innovation that runs mini-programs in the Linux kernel without modifying the kernel module. In eBPF, the execution of a program is triggered by the occurrence of an event. eBPF programs are run by attaching themselves to the predefined hooks such as system calls, kernel tracepoints, network events, etc. The users can also create their own probes which can be either kernel probe (kprobe) or user probe (uprobe). These probes can be attached to user applications, device drivers, system call interfaces, network sockets, etc., to trace the information in real-time. One such kprobe $tcp\_v6\_do\_rcv$ (for IPv6) can be used to trace TCP sockets that are active and collect socket related information such as source and destination IP addresses, source and destination ports, RTT, connection status, etc. One of the main advantages of eBPF over other kernel programs is that it is efficient and secure. Since it traces only TCP session events, which are less frequent, it creates a very low overhead and can be easily run for longer duration. This opens multiple possibilities, such as system monitoring, tracing, security, and network debugging.

1.3.5 Evolution of end devices in communication networks

The early-end devices were landline telephones and basic computers that had limited computation capabilities. These were primarily designed for voice communication and standalone processing. As personal computers emerged, they were able to

connect to LANs and share files and send emails. With the advent of mobile phones, wireless communication was possible, albeit with constrained data capacities in 2G. The introduction of 3G networks and smartphones brought about a significant advancement in end-device capabilities. These gadgets facilitated mobile web browsing, email, and messaging while on the go. They could also handle multimedia and execute programs. With the emergence of the IoT, a new wave of smart devices—such as wearables, smart home appliances, and sensors—was introduced. These network-connected gadgets gathered and shared real-time data to enable automation, control, and monitoring. Advanced features like local data processing via edge computing, which lowers latency and improves data rates, are now present in modern end devices. Smart cameras and industrial robots are examples of technologies that can integrate AI to evaluate data, carry out complicated tasks automatically, and facilitate real-time decision-making. End devices can utilize ultra-low latency, higher data rates, and greater bandwidth due to 5G and next-generation Wi-Fi technologies. This increases their ability to serve data-intensive applications such as AR/VR, autonomous systems, and expansive IoT ecosystems.

Though the modern end devices are getting stronger in terms of computational power and memory, some of them, with the added ability of analytics and independent decision-making capabilities, are never involved in the network decision-making. It is essential to investigate how and to what extent the end devices can be involved in collaborative decision-making achieving optimal network operation.

1.3.6 Reinforcement learning for communication networks

As mentioned earlier, the adoption of wireless technologies in private-professional networks, along with the diversity of applications in these networks, has increased over the years. As a result, future wireless networks will be incredibly complex, rendering conventional mathematical methods for network implementation, design, and operation insufficient. This has motivated the move towards data-driven approaches as elaborated in [53]. Communication networks are using Reinforcement Learning (RL) more frequently to improve performance and decision-making in dynamic, complicated situations. Within the field of RL, agents acquire the ability to make optimal decisions by trial and error, using input from the environment in the form of rewards. RL algorithms are being used to dynamically allocate bandwidth [54], spectrum [55], and power [56] in Wi-Fi and cellular networks to optimize performance in terms of throughput, latency, and adaptation to changing network conditions. They are also being used for intelligent traffic routing to learn the best routes to minimize congestion and delays. Some other areas of communication networking also use RL for network slicing [57] and load balancing [58]. Though these algorithms are adaptable and provide optimized solutions, they have very high complexities in terms of training, balance between exploration

and exploitation, and scalability.

1.4 Challenges

As described earlier, the increase of connected devices in private-professional networks increases the diversity of applications, which in turn increases the number of traffic flows in the network. These traffic flows have rigorous QoS requirements, which the private-professional networks in their current state cannot satisfy. Further, wider adoption of Wi-Fi technologies in these networks, poses additional challenges. Some of these challenges and gaps are listed below.

1.4.1 Existing CC algorithms are specific to wired networks

Challenge 1: Existing CC algorithms are network agnostic, performing decision-making based on limited end-to-end knowledge:

One of the advantages of TCP is the congestion-free communication utilizing its CC algorithm. The algorithm decides the number of application packets to be sent at each instant in time while avoiding congestion and maximizing data throughput. The algorithms operate based on partial end-to-end information available from the ACK packets received from the receiver device. The algorithms are also designed to leverage the high bandwidth capacity of wired networks. Whereas wireless networks pose additional challenges in terms of lossy medium, mobility, interference, variable data rates, etc. There have been techniques such as split connection approach [59], protocol conversion, link-level approach [60], [61], TCP protocol boosters [62], ECN [27], and several other explicit CC mechanisms that address the issue of congestion in wireless media, but with the disadvantage of not having detailed insight into the network. Specially ECN, when used in less crowded networks, can result in additional complexity without any benefits. With several applications moving towards wireless mediums, it is crucial to design a suitable CC algorithm that takes into account the wireless network conditions and challenges it poses to network congestion avoidance mechanisms.

1.4.2 Increased load on intermediate network nodes

Challenge 2: Existing intermediate network nodes are limited with the capabilities of incorporating new functionalities, specially to achieve fine grained flow treatment with increase in the number of flows.

Real-time network reconfiguration are possible through data plane programmability, application-network interaction, and advanced telemetry like INT (as shown in [47]), which allow for the collection and utilization of adequate information about network status and application requirements. However, there are obstacles to this

progress that must be overcome. In a W-TSN network, the difficulty of providing a fine-grained differentiated treatment, slicing, priority queuing, scheduling, etc., for every flow will soon hit its limit as the number of flows in the network rises, let's say to the hundreds or even thousands. More functionality such as traffic shaping, queuing, time-sensitivity, etc. is needed from the network components, which eventually exceeds their capacity to offer new features.

1.4.3 Fixed QoS options for wide range of applications

Challenge 3: Existing fixed set of QoS categories limit the diverse applications from choosing the right QoS category to achieve stringent requirements.

The applications running in private-professional wireless networks are diverse and impose quite heterogeneous and stringent requirements that must be satisfied by the underlying communication infrastructure, resulting in different traffic types demanding different QoS. Looking 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, achieving fine-grained DiffServ for each flow becomes harder. To overcome these issues, deterministic[4] and time-sensitive[5] 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 in a common treatment of flows with diverse needs. Additionally, there are challenges in terms of scalability, latency, and flexibility in centralized approaches while catering to these diverse and stringent demands that need to be addressed. Therefore there is a need to complement such mechanisms with decentralized approaches, able to adapt to the growing network traffic, at the same time considering the priorities of emerging applications and achieving congestion-free differentiation of services.

1.4.4 Throughput degradation in Wi-Fi networks due to airtime unfairness

Challenge 4: Airtime unfairness among end devices in a Wi-Fi network in the scenario of fair channel access results in throughput degradation in end devices with higher physical data rates.

A Wi-Fi network consists of an AP with multiple end devices connected to it. The physical data rate of the end device is determined by the channel quality

[4]Deterministic Networking: https://datatracker.ietf.org/wg/detnet/about/
[5]Time-Sensitive Networking Task Group: https://1.ieee802.org/tsn/

which varies based on the distance of the end device from the AP, obstacles in the environment, interference from other end devices, etc. Hence in a Wi-Fi network, each end device can have a different physical data rate based on the channel conditions. Since Wi-Fi uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), a pseudo-random fairness-based channel access method, in a mixed physical data rate network, the airtime consumed by the end device with a lower physical data rate is larger than that of an end device with higher physical data rate. This unfairness in airtime consumption slows down the end devices with higher physical data rates and degrades the performances of the entire network. There are mechanisms such as EDCA [63] mechanism, and the Transmit Opportunity (TXOP) feature of 802.11 to prioritize the demands of diverse applications. However, the airtime unfairness and throughput degradation due to a multitude of physical data rates across end devices are still evident for the same QoS class of EDCA. Similarly TXOP is supported only for a few access categories, i.e., background, best effort, video, and voice, and is applied to all the flows belonging to the same access category. But when the TXOP is enabled in all the end devices, the flows of the same access category end up facing airtime unfairness during the presence of an end device with a lower physical data rate. Therefore, in a mixed physical data rate Wi-Fi network, the throughput of all the end devices irrespective of their physical data rates is degraded by the presence of an end device with very low physical data rate [64–66]. Hence, it is essential to address the problem of throughput degradation in mixed physical data rate Wi-Fi networks, so that not all devices will suffer from it.

1.4.5 Summary of challenges

Based on these gaps, the challenges addressed in this thesis are summarized as follows,

1. The existing CC algorithms are designed for wired networks hence there are no suitable CC algorithms for wireless networks.

2. An increase in the number of end devices results in an increase in the number of traffic flows in the network components, which increases the network complexity and its management and the load on intermediate network nodes.

3. In the current state of the network, applications are confined to choosing QoS from fixed DiffServ, resulting in common treatment to wide range of applications.

4. Due to the fair channel access mechanism of Wi-Fi, the end devices with higher physical data rate experience degraded throughput because of airtime unfairness in the presence of devices with low physical data rates.

These challenges have been presented already in Figure 1.5

1.5 Research contributions

In Section 1.4, the problems and challenges in Wi-Fi-based private-professional networks are formulated. These challenges are tackled by studying the role of end devices, specially the transport layer CC algorithm. Various adaptive CC algorithms that utilize abundant network context from INT are designed. They are discussed in detail in the remainder of this PhD dissertation for which the outline is given in Section 1.6. To conclude, an elaborated list of the research contributions within this dissertation is presented as follows:

- Analyse the feasibility of adaptive transport layer.
  - Integration of eBPF with INT to simultaneously collect real-time TCP data with INT.
  - Study the relation between real-time network information from INT and $cwnd$.
  - Modify the existing CC algorithm to consider wireless packet loss and increase the window size based on the available queue capacity in the intermediate nodes.
  - Validate the performance of modified algorithm against the existing CC algorithm in Mininet-WiFi[6].
- Design of network- and Application-aware adaptive CC algorithm (NACC).
  - Design and implementation of application request processing in intermediate nodes.
  - Design and implementation of CC algorithm that provides congestion free differentiation in services based on application requirements and real-time network context.
  - Test the impact of real-time network parameters on the designed algorithm.
  - Benchmark the designed algorithm against the CUBIC [23] algorithm in terms of stability, throughput, fairness and application priorities in w-iLab.2 Testbed[7].
- Design of Reactive and Adaptive CC algorithm (RACC).

[6]Mininet-WiFi documentation: https://mn-wifi.readthedocs.io/en/latest/
[7]imec iLab.t testbeds' documentation: https://doc.ilabt.imec.be/ilabt/wilab/

  - Design of an adaptive CC algorithm that reacts to the airtime allocation feedback from the AP by modifying its application data transfer rate.
  - Benchmark the designed algorithm against the CUBIC algorithm in terms of airtime fairness and throughput in w-iLab.2 Testbed.

- Design of Feedback-based Control loop CC algorithm (FCCC).
  - Design of an aggregated airtime feedback system based on airtime usage in intermediate nodes.
  - Design of control-loop based CC algorithm that uses aggregated airtime information and real-time network context to achieve airtime fairness in wireless networks.
  - Benchmark the designed algorithm against the CUBIC algorithm in terms of airtime fairness and throughput in w-iLab.2 Testbed.
  - Evaluate the robustness of the designed algorithm to changing network conditions in w-iLab.2 Testbed.

- Design of multi-context-aware RL-based CC algorithm.
  - Model RL-based CC algorithm for wireless networks as contextual Markov decision process.
  - Design of three different multi-context-aware RL-based CC algorithm for private wireless networks.
  - Implement, train and test two of the proposed three methods using Epsilon-Greedy Q learning in w-iLab.2 Testbed.

The designed CC algorithms provide congestion-free data transfer service in Wi-Fi networks by utilizing the rich network insights in real-time. In addition to providing congestion-free services, the designed algorithms facilitate airtime fairness, application requirement fulfillment, and adapting to changing network conditions in Wi-Fi networks. The algorithms were implemented on commercially available Wi-Fi devices and were validated against the existing CUBIC CC algorithm. The better performance of designed algorithms in terms of QoS, airtime fairness, and adaptability to wireless conditions assures the importance of real-time network insights in higher-layer protocols.

1.6 Outline

This dissertation is composed of a number of publications that were realized within the scope of this PhD. The selected publications provide an integral and consistent overview of the work performed. The different research contributions are detailed

in the Section 1.5 and the complete list of publications that resulted from this work is presented in Section 1.7. Within this section an overview of the remainder of this dissertation is given and how the different chapters are linked together is explained. Table 1.1 shows the challenges that were highlighted in Section 1.4 and indicates which were targeted per chapter.

To design a CC algorithm that considers the challenges of wireless networks, firstly Chapter 2 analyzes the relation between INT parameters and $cwnd$ by integrating eBPF with INT. By utilizing the real-time INT information, the shows the feasibility of adaptive transport layer protocols to address the challenges of future private-professional networks. Based on the study in Chapter 2, Chapter 3 discusses and validates the design of the network-application-aware adaptive CC algorithm. This algorithm addresses the issue of fixed QoS for diverse applications by considering the application requirements in decision-making. Additionally, the algorithm is suitable for wireless network conditions as it uses real-time INT information to decide $cwnd$. The entire system is decentralized, and decision-making is being moved to the end devices, which further reduces the complexity of intermediate network nodes.

The challenge of airtime unfairness in Wi-Fi networks is addressed by first designing a reactive CC algorithm in Chapter 4 and extending it to a feedback-based control loop CC algorithm, which is proactive in nature, in Chapter 5. To address the challenges of increasing complexity of future private-professional networks and diversity of applications, a data-driven solution is approached in Chapter 6. The chapter proposes three different multi-context-aware RL-based CC solutions for wireless networks, of which two are validated. The solutions consider the unpredictability of wireless conditions by adding the channel conditions as contexts.

Appendix A gives the description of the application-network interaction API used in designing all the CC algorithms presented in the thesis.

Table 1.1: An overview of the contributions per chapter in this dissertation.

Challenges	Ch.2	Ch.3	Ch.4	Ch.5	Ch.6
Existing CC algorithms are specific to wired networks	•	•	•	•	•
Increased load on intermediate network nodes in private-professional wireless networks		•	•	•	•
Fixed QoS options for wide range of applications		•			
Throughput degradation in Wi-Fi networks due to airtime unfairness			•	•	

1.7 Publications

The research results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences. The following list provides an overview of the publications during the PhD research.

1.7.1 Publications in international journals (listed in the Science Citation Index)

1. **Ramyashree Venkatesh Bhat**, Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke. *Network- and Application-Aware Adaptive Congestion Control Algorithm.* Published in Journal of Communications and Networks, 26(3), 344–355, 2024.

2. **Ramyashree Venkatesh Bhat**, Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke. *Feedback-Based Control Loop Congestion Control Algorithm for Wireless Networks.* Published in IEEE Access 12: 101767–81, 2024.

1.7.2 Publications in international conferences (listed in the Science Citation Index)

1. Jetmir Haxhibeqiri, Amina Seferagic, **Ramyashree Venkatesh Bhat**, Ingrid Moerman, and Jeroen Hoebeke. *Tighter Application-Network Interfacing to Drive Innovation in Networked Systems.* Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration (NAI '21), Association for Computing Machinery (ACM), 2021, pp. 53–57.

2. Jetmir Haxhibeqiri, **Ramyashree Venkatesh Bhat**, Ingrid Moerman, and Jeroen Hoebeke. *Age-of-Information Aware in-Band Network Telemetry for Better Network Predictability.* 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE, 2021, pp. 42–47.

3. **Ramyashree Venkatesh Bhat**, Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke. *Adaptive Transport Layer Protocols Using In-Band Network Telemetry and EBPF.* Published in 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE, 2021, pp. 241–46.

4. **Ramyashree Venkatesh Bhat**, Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke. *In-band Network Telemetry-based Congestion Control Algorithm for Industrial Wireless Networks* Published in 2024 IEEE 29TH

International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2024.

5. **Ramyashree Venkatesh Bhat**, Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke. *Multi-Context-aware RL approach towards INT-based Congestion Control Algorithm.* Accepted in 2024 IEEE Conference on Standards for Communications and Networking, IEEE, 2024.

References

[1] N. Finn. *Introduction to time-sensitive networking*. IEEE Communications Standards Magazine, 2(2):22–28, 2018.

[2] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek. *Timely survey of time-sensitive networking: Past and future directions*. Ieee Access, 9:142506–142527, 2021.

[3] S. Rodrigues and J. Lv. *Synchronization in Time-Sensitive Networking: An Introduction to IEEE Std 802.1 AS*. IEEE Communications Standards Magazine, 6(4):14–20, 2022.

[4] S. R. P. SRP. *Bridges and Bridged Networks*.

[5] I. . documentation. *Credit-Based Shaping*. https://inet.omnetpp.org/docs/showcases/tsn/trafficshaping/creditbasedshaper/doc/index.html, 2024. [Online; accessed 06-January-2024].

[6] I. . documentation. *Time-Aware Shaping*. ttps://inet.omnetpp.org/docs/showcases/tsn/trafficshaping/timeawareshaper/doc/index.html, 2024. [Online; accessed 06-January-2024].

[7] I. . documentation. *Asynchronous Traffic Shaping*. https://inet.omnetpp.org/docs/showcases/tsn/trafficshaping/asynchronousshaper/doc/index.html, 2024. [Online; accessed 06-January-2024].

[8] F. A. Mahamid and K. Holzinger. *"time sensitive networking-802.1 qci*. Network, 9, 2021.

[9] J. Ahmad. *Assessment of Industrial Networking Technologies for Support of Time-Critical Communication*. Master's thesis, 2023.

[10] D. Cavalcanti, C. Cordeiro, M. Smith, and A. Regev. *WiFi TSN: Enabling deterministic wireless connectivity over 802.11*. IEEE Communications Standards Magazine, 6(4):22–29, 2022.

[11] H. Chahed and A. Kassler. *TSN Network Scheduling—Challenges and Approaches*. Network, 3(4):585–624, 2023.

[12] L. Galati-Giordano, G. Geraci, M. Carrascosa, and B. Bellalta. *What will Wi-Fi 8 be? A primer on IEEE 802.11 bn ultra high reliability*. IEEE Communications Magazine, 62(8):126–132, 2024.

[13] J. Haxhibeqiri, X. Jiao, X. Shen, C. Pan, X. Jiang, J. Hoebeke, and I. Moerman. *Coordinated SR and restricted TWT for time sensitive applications in WiFi 7 networks*. IEEE Communications Magazine, 62(8):118–124, 2024.

[14] P. Imputato and S. Avallone. *Meeting Latency Constraints in Wi-Fi Through Coordinated OFDMA*. In 2024 22nd Mediterranean Communication and Computer Networking Conference (MedComNet), pages 1–4. IEEE, 2024.

[15] D. Bravo, L. Cariou, T. Kenney, and R. Stacey. *Preemption mechanism for wlan*, March 18 2021. US Patent App. 17/109,223.

[16] E. Reshef and C. Cordeiro. *Future directions for Wi-Fi 8 and beyond*. IEEE Communications Magazine, 60(10):50–55, 2022.

[17] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman. *openwifi: a free and open-source IEEE802. 11 SDR implementation on SoC*. In 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), pages 1–2. IEEE, 2020.

[18] M. Aslam, W. Liu, X. Jiao, J. Haxhibeqiri, J. Hoebeke, I. Moerman, E. Municio, P. Isolani, G. Miranda, and J. Marquez-Barja. *High precision time synchronization on Wi-Fi based multi-hop network*. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 1–2. IEEE, 2021.

[19] P. Avila-Campos, J. Haxhibeqiri, I. Moerman, X. Jiao, and J. Hoebeke. *Impactless association methods for wi-fi based time-sensitive networks*. Wireless Networks, pages 1–19, 2024.

[20] P. Avila-Campos, J. Haxhibeqiri, X. Jiao, B. Van Herbruggen, I. Moerman, and J. Hoebeke. *Unlocking mobility for wi-fi-based wireless time-sensitive networks*. IEEE Access, 2024.

[21] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. *The quic transport protocol: Design and internet-scale deployment*. In Proceedings of the conference of the ACM special interest group on data communication, pages 183–196, 2017.

[22] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. *The NewReno modification to TCP's fast recovery algorithm*. Technical report, 2012.

[23] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *CUBIC for fast long-distance networks*. Technical report, 2018.

[24] S. Ha, I. Rhee, and L. Xu. *CUBIC: a new TCP-friendly high-speed TCP variant*. ACM SIGOPS operating systems review, 42(5):64–74, 2008.

[25] T. Kato, S. Haruyama, R. Yamamoto, and S. Ohzahata. *mpCUBIC: A CUBIC-like congestion control algorithm for multipath TCP*. In Trends and Innovations in Information Systems and Technologies: Volume 2 8, pages 306–317. Springer, 2020.

[26] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. *BBR: Congestion-Based Congestion Control*. ACM Queue, 14, September-October:20 – 53, 2016. Available from: http://queue.acm.org/detail.cfm?id=3022184.

[27] S. Floyd. *TCP and explicit congestion notification*. ACM SIGCOMM Computer Communication Review, 24(5):8–23, 1994.

[28] K. De Schepper and G. White. *RFC 9332: Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)*, 2023.

[29] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines. *5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges*. Computer Networks, 167:106984, 2020.

[30] D. Harrington, R. Presuhn, and B. Wijnen. *An architecture for describing simple network management protocol (SNMP) management frameworks*. Technical report, 2002.

[31] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann. *OpenSDWN: Programmatic control over home and enterprise WiFi*. In Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research, pages 1–12, 2015.

[32] F. Paolucci, F. Cugini, P. Castoldi, and T. Osiński. *Enhancing 5G SDN/NFV edge with P4 data plane programmability*. IEEE Network, 35(3):154–160, 2021.

[33] C. Metter, V. Burger, Z. Hu, K. Pei, and F. Wamser. *Towards an active probing extension for the ONOS SDN controller*. In 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), pages 1–8. IEEE, 2018.

[34] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers. *Opennetmon: Network monitoring in openflow software-defined networks*. In 2014 IEEE Network Operations and Management Symposium (NOMS), pages 1–8. IEEE, 2014.

[35] S. Shirali-Shahreza and Y. Ganjali. *Traffic statistics collection with FleXam*. ACM SIGCOMM Computer Communication Review, 44(4):117–118, 2014.

[36] B. Claise. *Cisco systems netflow services export version 9*. Technical report, 2004.

[37] B. Claise. *Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information*. Technical report, 2008.

[38] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L. J. Wobker, et al. *In-band network telemetry via programmable dataplanes*. In ACM SIGCOMM, volume 15, pages 1–2, 2015.

[39] A. Gulenko, M. Wallschläger, and O. Kao. *A practical implementation of in-band network telemetry in open vswitch*. In 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pages 1–4. IEEE, 2018.

[40] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong. *Intcollector: A high-performance collector for in-band network telemetry*. In 2018 14th International Conference on Network and Service Management (CNSM), pages 10–18. IEEE, 2018.

[41] N. Van Tu, J. Hyun, and J. W.-K. Hong. *Towards ONOS-based SDN monitoring using in-band network telemetry*. In 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), pages 76–81. IEEE, 2017.

[42] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li. *In-band network telemetry: A survey*. Computer Networks, 186:107763, 2021.

[43] A. Karaagac, E. De Poorter, and J. Hoebeke. *Alternate marking-based network telemetry for industrial WSNs*. In 2020 16th IEEE International Conference on Factory Communication Systems (WFCS), pages 1–8. IEEE, 2020.

[44] A. Karaagac, E. De Poorter, and J. Hoebeke. *In-band network telemetry in industrial wireless sensor networks*. IEEE Transactions on Network and Service Management, 17(1):517–531, 2019.

[45] P. H. Isolani, J. Haxhibeqiri, I. Moerman, J. Hoebeke, J. M. Marquez-Barja, L. Z. Granville, and S. Latré. *An SDN-based framework for slice orchestration using in-band network telemetry in IEEE 802.11*. In 2020 6th IEEE Conference on network softwarization (NetSoft), pages 344–346. IEEE, 2020.

[46] J. Haxhibeqiri, I. Moerman, and J. Hoebeke. *Low overhead, fine-grained end-to-end monitoring of wireless networks using in-band telemetry*. In 2019 15th international conference on network and service management (CNSM), pages 1–5. IEEE, 2019.

[47] 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 Transactions on Network and Service Management, 18(1):627–641, 2020.

[48] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann. *Socket intents: Leveraging application awareness for multi-access connectivity*. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, pages 295–300, 2013.

[49] B. Hesmans and O. Bonaventure. *An enhanced socket API for Multipath TCP*. In Proceedings of the 2016 applied networking research workshop, pages 1–6, 2016.

[50] K. R. Evensen, K.-J. Grinnemo, A. F. Hansen, N. Khademi, S. Mangiante, P. McManus, G. Papastergiou, D. Ros, M. Tüxen, E. Vyncke, et al. *The NEAT Architecture*. Online) https://www. neat-project. org, 2015.

[51] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke. *Tighter application-network interfacing to drive innovation in networked systems*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 53–57, 2021.

[52] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal. *Creating complex network services with ebpf: Experience and lessons learned*. In 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), pages 1–8. IEEE, 2018.

[53] A. Zappone, M. Di Renzo, and M. Debbah. *Wireless networks design in the era of deep learning: Model-based, AI-based, or both?* IEEE Transactions on Communications, 67(10):7331–7376, 2019.

[54] L. Ruan, M. P. I. Dias, and E. Wong. *Towards self-adaptive bandwidth allocation for low-latency communications with reinforcement learning*. Optical Switching and Networking, 37:100567, 2020.

[55] W. Lei, Y. Ye, and M. Xiao. *Deep reinforcement learning-based spectrum allocation in integrated access and backhaul networks*. IEEE Transactions on Cognitive Communications and Networking, 6(3):970–979, 2020.

[56] H. Yang, X. Xie, and M. Kadoch. *Intelligent resource management based on reinforcement learning for ultra-reliable and low-latency IoV communication networks*. IEEE Transactions on Vehicular Technology, 68(5):4157–4169, 2019.

[57] J. A. Hurtado Sánchez, K. Casilimas, and O. M. Caicedo Rendon. *Deep reinforcement learning for resource management on network slicing: A survey*. Sensors, 22(8):3031, 2022.

[58] D. Wu, J. Kang, Y. T. Xu, H. Li, J. Li, X. Chen, D. Rivkin, M. Jenkin, T. Lee, I. Park, et al. *Load balancing for communication networks via data-efficient deep reinforcement learning*. In 2021 IEEE global communications conference (GLOBECOM), pages 01–07. IEEE, 2021.

[59] A. Bakre and B. Badrinath. *I-TCP: Indirect TCP for mobile hosts*. In Proceedings of 15th International Conference on Distributed Computing Systems, pages 136–143. IEEE, 1995.

[60] H. B. (hari@lcs.mit.edu). *The Berkeley Snoop Protocol* . http://nms.lcs.mit.edu/~hari/papers/snoop.html. [Online; accessed 06-January-2024].

[61] H. B. (hari@lcs.mit.edu). *A technical tutorial on the 802.11 standard*. http://www.sss-mag.com/pdf/802_11tut.pdf, 1997. [Online; accessed 06-January-2024].

[62] J. Hoebeke, T. Van Leeuwen, L. Peters, K. Cooreman, I. Moerman, B. Dhoedt, and P. Demeester. *Development of a TCP protocol booster over a wireless link.* In Proceedings of the 9th Symposium on Communications and Vehicular Technology in the Benelux (SCVT 2002), organised by the IEEE Benelux Chapter on Communications and Vehicular Technology, October 17, 2002, Louvain-La-Neuve, Belgium, pages 27–34, 2002.

[63] F. Peng, B. Peng, and D. Qian. *Performance analysis of IEEE 802.11 e enhanced distributed channel access*. IET communications, 4(6):728–738, 2010.

[64] A. M. Abdul-Hadi, O. Tarasyuk, A. Gorbenko, V. Kharchenko, and T. Hollstein. *Throughput estimation with regard to airtime consumption unfairness in mixed data rate Wi-Fi networks*. Communications-Scientific letters of the University of Zilina, 16(1):84–89, 2014.

[65] F. Safdari and A. Gorbenko. *Experimental Evaluation of Performance Anomaly in Mixed Data Rate IEEE802. 11ac Wireless Networks*. In 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT), pages 82–87. IEEE, 2019.

[66] F. Safdari and A. Gorbenko. *Theoretical and experimental study of performance anomaly in multi-rate IEEE802. 11ac wireless networks*. Radioelectronic and Computer Systems, (4):85–97, 2022.

2

Feasibility of adaptive transport layer protocols

This chapter examines the possibility of adaptive transport layer protocols by determining the relationship between the $cwnd$ and real-time network parameters from INT. TCP data and INT information are synchronized through the integration of eBPF with INT. The study's findings showed that the behavior of the $cwnd$ directly affects the important variables of intermediate network nodes, including the number of flows, buffer queue capacity, and packet arrival rate. The current CUBIC CC method was adjusted to take packet loss resulting from wireless link circumstances into account using the INT information. The improved algorithmic performance proved the possibility of comprehensive understanding of the network's condition.

⋆ ⋆ ⋆

This chapter is based on:
Adaptive Transport Layer Protocols using In-band Network Telemetry and eBPF

Published in WiMob, October 2021.

⋆ ⋆ ⋆

2.1 Introduction

The transport layer of the OSI model provides a logical host-to-host communication service. TCP and UDP are the two popularly used transport layer protocols, with TCP differing from UDP because of the ordered, reliable, error-free data transfer, flow control, and CC mechanisms. These mechanisms operate based on partial end-to-end information available from the ACK packets received from the other end. Existing CC algorithms have been particularly designed for wired networks. With several applications moving towards wireless media, it is crucial to design a suitable CC algorithm that takes into account the wireless network conditions such as lossy medium, mobility, thick walls, etc.

Recently there has been progress in the area of application-network interactions [1] for easy extensibility and on-the-fly customization of higher-layer protocols, as well as in the area of network monitoring and verification. INT, initially implemented for wired networks, has been extended to wireless networks by designing an architecture and logic to generate and process INT-enabled packets for Wi-Fi-based networks [2]. This design has a low overhead and is capable of collecting real-time information on end-to-end, per-hop and per-flow basis. Considering this, there is a great opportunity for optimizing the behavior of transport protocols by exploiting the in-depth insights that can be acquired from INT in their decision-making. To validate this opportunity, this chapter focuses on the TCP CC mechanism behaviour, by adjusting the $cwnd$, a key variable limiting the data transfer, based on the real-time network information obtained from INT.

To achieve this, it is necessary to first understand the relation between CC algorithm behavior and different monitored INT parameters for wireless networks. The recent technology, eBPF, provides an option to run mini-programs in a safe virtual machine in the Linux kernel with low overhead and can be used to monitor TCP sockets in real-time. In this paper,

- We integrate eBPF with INT to collect real-time data for both TCP behavior and network parameters.
- From this, we can derive a relationship between INT parameters and TCP parameter changes, e.g. TCP $cwnd$.
- Such derived relation is used to modify in real-time the $cwnd$ for optimizing the transport layer behavior for different network scenarios.

The rest of the chapter is structured as follows, Section 2.2 provides information about existing works in the area of INT and eBPF. A brief introduction on the problems in wireless networks is provided in Section 2.3. Section 2.4 gives a detailed description of the problem statement. Section 2.5 and Section 2.6 present the methodologies and implementations used to address the problem statements

along with the obtained results and its discussion. Finally, the chapter is concluded in Section 2.7.

2.2 Related work

This section discusses about different research works in the area of eBPF.

Extended Berkeley Packet Filter

The eBPF is a pioneering innovation that can run programs in the Linux kernel without changing the kernel source code or loading the kernel modules or affecting other applications and systems [3]. It has been mainly used for system monitoring purposes, but it can also be used for tracing, security, and networking applications. In-network programmability based on IPv6 segment routing using eBPF is presented in [4]. An extensible Linux TCP stack, with new eBPF callbacks to support user-defined TCP options is implemented in [5]. In combination with P4, eBPF has been used to implement INT in OpenVSwitch (OVS) [6].

The new CC designs introduced in this thesis are based on INT and eBPF, both with very low overheads, whereas the existing explicit CC techniques require additional bits in each packet, are moderately expensive in routers, and require modification of all the intermediate routers and switches. Also, INT gives real-time network updates, hence faster adaption to the network changes.

2.3 Problems in wireless networks

In general wireless networks face several problems with connectivity, security, network expansion, interference, etc., and network congestion being one of them. Network congestion occurs when a buffer is overflown in an intermediate node due to multiple devices using the same network path, or misconfiguration, or a large amount of data transfer. Network congestion causes queuing delay, packet loss, or blocking of new connections, thus reducing the network quality in terms of signal strength and physical data rates. Apart from this, wireless networks also face packet losses due to wireless medium characteristics (interference, long distance between device, mobility, thick walls etc). The TCP protocol is designed to tackle packet losses due to network congestion only and is thus relevant to wired networks. Since the same mechanisms are used in wireless networks, these mechanisms generalize each packet loss as a consequence of network congestion, thus reducing the packet transfer rate for every packet loss and resulting in poorer data transfer in wireless networks.

2.4 Problem statement

The advancement in the field of application-network interaction [7] and the innovations to obtain real-time network information on a per-hop basis paves a way for extensibility and customization of transport layer network protocols. Also, as discussed in Section 2.3, the existing CC algorithms are more suitable for wired networks, and there is a requirement for a better CC mechanism for wireless scenarios. We focus on adapting the existing CC mechanisms of TCP to the wireless network conditions, such as packet losses due to interference or mobility, physical data rate, etc., based on the real-time network information obtained from the INT. To achieve this, we try to answer the following questions,

1. Is there a relationship between the parameters obtained from INT and the $cwnd$ of TCP?
2. If so, can we use the INT information to modify the $cwnd$ in real-time?
3. Does the new technique improve the performance of the TCP for wireless network conditions?

2.5 TCP and INT Parameters Relation

In this section, we derive the relation between the INT and TCP parameters related to CC algorithm.

2.5.1 Collecting the data points for analysis

To find the relation between the INT data and the $cwnd$, it is essential to measure both INT and $cwnd$ for each packet. For wireless networks, the INT header is encapsulated as an IPv6 option [2]. To collect the INT data and $cwnd$ at the same time, we integrate the real-time TCP data traced by eBPF in the INT end-to-end option, as shown in Figure 2.1. Other network parameters, such as queue filling, timestamping, packet losses and wireless link parameters (RSSI, data rate, MCS) are still included as hop-by-hop option in INT data, but are not shown in Figure 2.1. We also add a TCP flag (which is set to 1 when the data traced using eBPF is included in the extension header) to indicate the presence of real-time TCP data to the sink. All the alignment in the INT extension is done as 4 bytes as required.

2.5.2 Implementation

As it is difficult to directly code in eBPF, a tool called BPF Compiler Collection (BCC) provides front-ends in Python and Lua to write BPF programs with

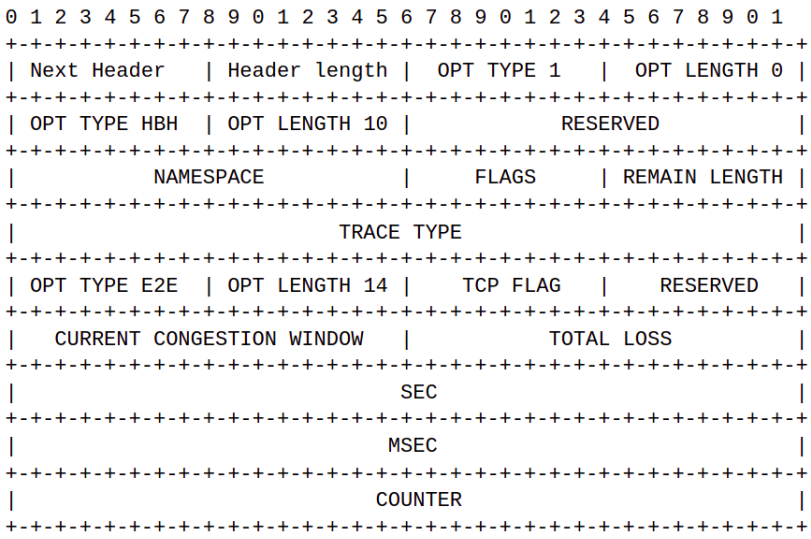


Figure 2.1: Integration of eBPF traced data with INT as IPv6 extension header

kernel instrumentation in C, including a C wrapper around LLVM [1] The INT and INT-enabled node architecture is implemented in the Click modular router framework in [2]. We integrate the INT-enabled node architecture with the BCC library to trace the TCP data in real-time using eBPF. The INT source node adds the real-time $cwnd$ as an end-to-end parameter inside the INT data structure, along with the INT header. The data is collected for different network scenarios and the results obtained are explained in the next subsection.

2.5.3 Test Setup and Results

The proposed design and implementation are validated on a multi-AP network setup in Mininet-WiFi[2] with two APs (AP1 and AP2) operating at a 2.4 GHz frequency, i.e., using Wi-Fi IEEE 802.11g, and two end devices (End device 1 and End device 2) as shown in Figure 2.2. All the devices use Ubuntu 20.04 as the operating system, with kernel verison 5.4.289, TCP version as per the standard in [8] and IP version 6. Each interface is assigned an IPv6 address with each of the nodes running the INT-enabled architecture for network monitoring. In this section, we discuss the relationship between the INT parameters and the $cwnd$.

To collect sufficient amounts of data, INT along with eBPF traced TCP data is collected for each packet, for a fixed amount of data bytes in iperf3. The data

[1] BCC - Tools for BPF-based Linux IO: https://github.com/iovisor/bcc
[2] Mininet-WiFi documentation: https://mn-wifi.readthedocs.io/en/latest/

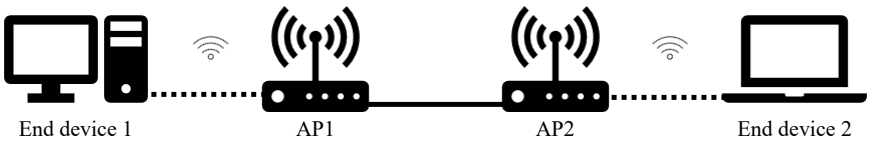


Figure 2.2: Multi-AP network setup in Mininet-WiFi

was collected for different network conditions by introducing intentional loss in a wireless link and delay with bounded queue capacity in the intermediate nodes for the CUBIC algorithm. The recorded data points were analyzed over time to understand the correlation between the two parameters. The key outcomes are summarized in the graphs in Figure 2.3 and Figure 2.4.

From the graph in Figure 2.3 it can be seen that after every loss event, there is a reduction in the $cwnd$. These loss events are detected by INT and are due to the lossy wireless medium. The higher the number of losses, the higher the reduction in the window size. Hence they are inversely proportional to each other. The second graph in Figure 2.4 shows the queue filling or packets in the queue in an intermediate node with bounded queue capacity (queue capacity is bounded to 50 packets) and the $cwnd$ at the source over time. When buffer space is available in the intermediate node, the $cwnd$ starts increasing slowly. The buffer space gets filled and the number of packets in the queue keeps increasing. The CC algorithm, being unaware of the situation in intermediate nodes, keeps increasing the $cwnd$ unless there is a packet loss due to the buffer overflow.

In the Figure 2.4 between the points 12 s and 16 s, we can clearly see how increasing the $cwnd$ affects the buffer space in intermediate node. Therefore, the $cwnd$ increase can be related to the available queue capacity left in the intermediate nodes, that can easily be tracked by INT data. Similarly, an increase in the $cwnd$ increases the packet arrival rate at intermediate nodes, that can as well be easily tracked by the INT data.

To answer the questions 1 and 2 from Section 2.4, there is a clear relation between the $cwnd$ and the parameters such as available queue space and packet loss events that are obtained from INT data. As such, these parameters can be used to predict and modify the $cwnd$.

2.6 INT-based adaptive TCP

Based on the previous section's insights regarding the relation between INT and TCP parameters, in this section, we propose and validate the adaptive CC algorithm for TCP based on INT monitored parameters.

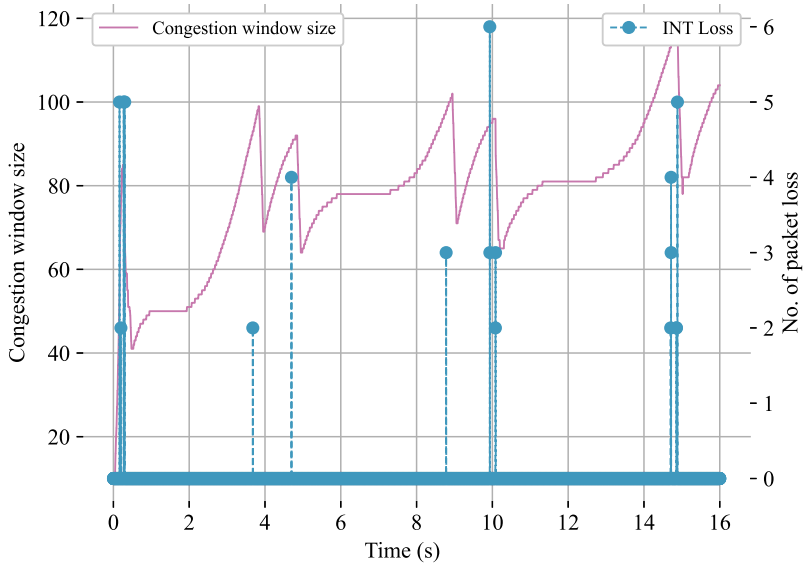


Figure 2.3: Relation between packet losses detected by INT and cwnd

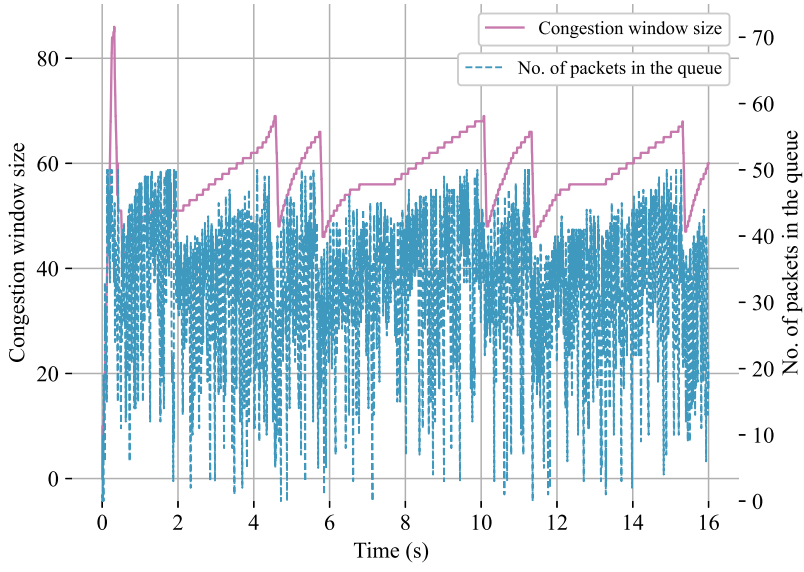


Figure 2.4: Relation between the no. of packets in queue and cwnd

Table 2.1: Behavior of new CC algorithms on packet loss due to lossy medium

	β	$cwnd$ **at loss event**
CUBIC New L1	-	$cwnd$ is kept constant
CUBIC New L2	0.95	$cwnd$ is decreased

2.6.1 Modifying the CC algorithm based on INT

The INT implementation in [2] collects intermediate node characteristics such as available queue capacity, processing delay, packet arrival rate, flow count, and Tx/Rx timestamping values, along with wireless link information (such as data rate, RSSI, SNR, the channel used, etc.) and end-to-end flow characteristics such as flow latency, flow jitter, and flow packet loss ratio. Since INT collects the detailed information of the intermediate nodes, it can easily differentiate between the packet losses due to buffer overflow and lossy wireless medium based on the flow packet loss ratio and the available queue capacity. The conventional CC algorithm reduces the window size irrespective of the reason for the packet loss. Whenever the packet loss is due to the lossy medium, we design two different techniques to maintain the throughput,

1. To keep the $cwnd$ constant: We implement a new function where on every packet loss due to the lossy medium, the $cwnd$ is kept constant.

2. To reduce the percentage of multiplicative decrease of the window size: On packet loss, CUBIC Linux decreases the window size multiplicatively by a factor of β, where β is a window decrease constant set to 0.7 [9]. In our design, for every packet loss which is not due to network congestion, the β value of 0.95 is used.

CUBIC New L1 and CUBIC New L2 are the algorithms designed to address the issue of packet losses in wireless networks due to reasons other than network congestion, whose parameters are shown in Table 2.1.

The available queue capacity, packet arrival rate, and flow count information of intermediate nodes indicate the business of the network and the buffer capacity of the node. This information is useful to decide the amount of data transfer at the sender side to avoid congestion in the node. From the result in Figure 2.4 it is evident that $cwnd$ is directly proportional to available queue capacity (q) and inversely proportional to the packet arrival rate (a_{packet}) and number of flows (fc) in the intermediate node. On every ACK packet, if the $cwnd$ is less than the $ssthresh$, the conventional CC algorithm increases the window size in steps of one. In our design, we utilize the information obtained from INT and increase the

window size based on that. We calculate the $cwnd$ based on the following formula,

$$cwnd = cwnd + \frac{(q * increment)}{(a_{packet} * fc)} \tag{2.1}$$

where $increment$ is a variable that decides the factor by which $cwnd$ is increased after every ACK. The modifications are represented in Algorithm 1.

Algorithm 1 Modifications to "On each ACK()" function in CUBIC algorithm in [10]

1: extract INT information from ACK such as, q, a_packet, fc
2: Follow the steps as mentioned in [10]
3: instead of incrementing in steps of 1, $cwnd \leftarrow cwnd + \frac{(q*increment)}{(a_{packet}*fc)}$

2.6.2 Implementation

To modify the $cwnd$ of Linux in real-time, we use the CC Plane (CCP) library [11]. CCP is an API by MIT which offers a separate plane for CC algorithms, thus enabling the option to write our CC methods. The CUBIC algorithm used in Linux is implemented in Python using CCP, which subscribes to a central broker to receive the real-time INT information. Each of the above-mentioned designs is integrated into the existing CUBIC algorithm separately and tested to compare the throughput with the original algorithm.

2.6.3 Results and discussion

The proposed design and implementation are validated on the same setup as shown in Figure 2.2. In this subsection, we firstly present the overall end-to-end performance of the new CC algorithm design based on INT data under wireless network loss conditions. Lastly, we discuss the progression of $cwnd$, especially during the slow start phase, by comparing it between our proposed enhanced CC algorithm and the CUBIC algorithm.

2.6.3.1 End-to-end performance of the new designs

In Section 2.6.1, we presented two different techniques to obtain $cwnd$ for a wireless medium when the packet losses are not due to network congestion. These techniques were tested separately on the emulated setup and the end-to-end performance was measured to compare them with the original CUBIC algorithm. Figure 2.5 indicates the throughput comparison between the existing TCP CUBIC in Linux and the new designs, CUBIC New L1 and CUBIC New L2, designed to address the issue of packet losses in wireless networks due to reasons other than network congestion,

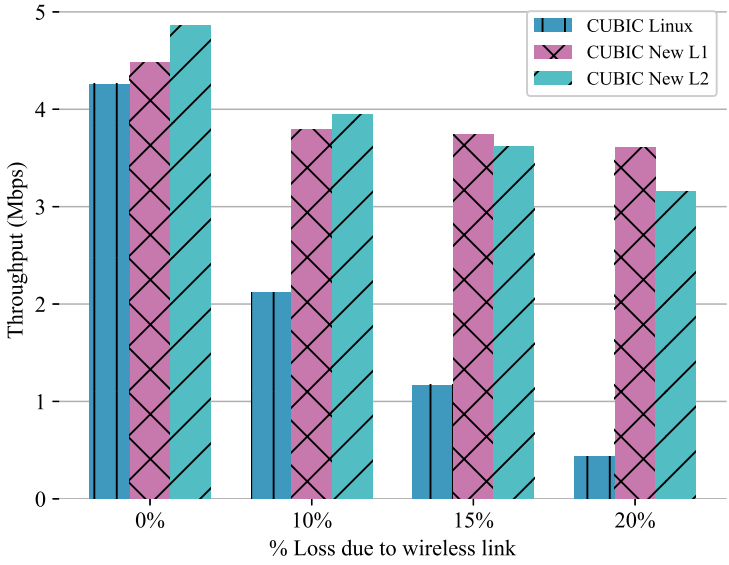


Figure 2.5: Comparison of original CUBIC algorithm with the proposed algorithms

whose parameters are shown in Table 2.1. The throughput values were measured for wireless medium with different loss percentages. From the graph, it can be seen that the new algorithms are capable of maintaining better throughput values even with the higher link loss percentages. In the case of a wireless link with a 20% packet loss ratio, the new designs using the real-time INT data achieve seven times higher throughput than the default Linux TCP congestion mechanism. Even for the scenario without any wireless link loss, CUBIC New L2 performs 12% better than the existing CUBIC algorithm.

2.6.3.2 Progression of $cwnd$

Using the design in Equation 2.1, we increase the $cwnd$ based on the information obtained from the intermediate nodes using INT. The graph in Figure 2.6 indicates the evolution of the $cwnd$ over time for $increment$ values of 1 (CUBIC New 1), 2 (CUBIC New 2), and 3 (CUBIC New 3) and compared with the original CUBIC algorithm (CUBIC Linux) implemented in Linux.

From the graph Figure 2.6, it can be noticed that for the algorithms with $increment$ values 2 and 3, the $cwnd$ reaches $ssthresh$ way before the original CUBIC algorithm. All the newer algorithms quickly reach the optimal value based on the intermediate node conditions and maintain the same $cwnd$ throughout the data transfer (especially the algorithm with increment = 1). For wireless network scenario, we prefer the algorithms with $increment$ values 2 and 3. They

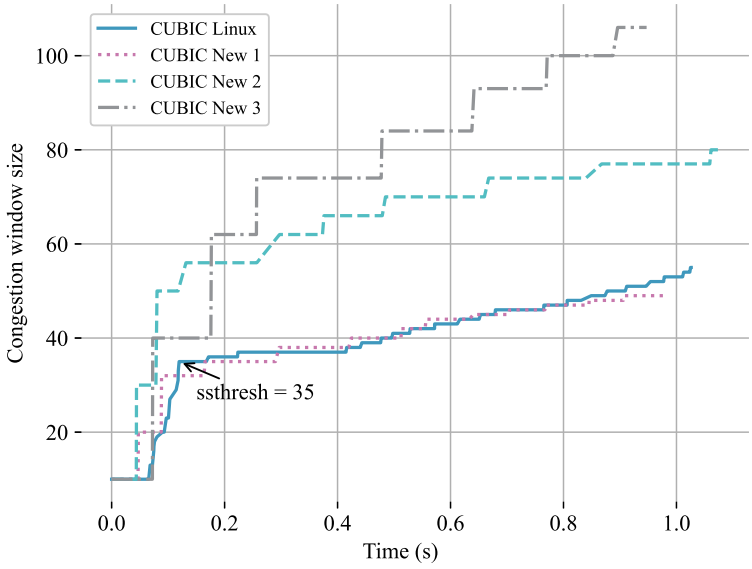


Figure 2.6: Progression of cwnd over time for different algorithms

transfer the data consistently over time, with better throughput and lower packet retransmissions.

To answer the question from Section 2.4, the designed algorithms were tested for different network conditions and the results were compared with the original CUBIC algorithm. When the intentional loss was introduced in the network, the designed algorithms not only maintained the throughput, but also achieved higher throughput than the original one. The algorithm designed using Equation 2.1 showed consistency in the data transfer rates and lower packet retransmissions.

Finally to summarise all the answers, from the conducted tests, we were able to correlate between the data obtained from INT and the $cwnd$ of TCP. Using the recent revolution in the Linux kernel called eBPF, we could modify the $cwnd$ in real-time. The newly designed techniques were tested for different network scenarios. There was an improvement in the overall network performance as compared to the original CUBIC algorithm, especially for a wireless network with a lossy medium.

2.7 Conclusion

The current TCP behavior is mainly based on the partial information obtained from the TCP ACKs, which leads to suboptimal decisions, particularly in wireless settings. In this chapter, the innovation of network monitoring and verification were utilized in designing newer algorithms for CC in wireless networks. With this we

could achieve up to seven times better throughput than the existing algorithm under 20% wireless link loss condition. This low overhead design helps the sender to adapt to the changes in the network, thus improving the overall network performance. It provides flexibility to change the data transfer parameters in real-time, allowing designing new transport protocols which can adjust based on the application requirements, resulting in tighter interaction between the protocol and the network.

References

[1] D. Lachos, Q. Xiang, C. Rothenberg, S. Randriamasy, L. M. Contreras, and B. Ohlman. *Towards deep network & application integration: Possibilities, challenges, and research directions*. In Proceedings of the Workshop on Network Application Integration/CoDesign, pages 1–7, 2020.

[2] 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 Transactions on Network and Service Management, 18(1):627–641, 2020.

[3] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal. *Creating complex network services with ebpf: Experience and lessons learned*. In 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), pages 1–8. IEEE, 2018.

[4] M. Xhonneux, F. Duchene, and O. Bonaventure. *Leveraging ebpf for programmable network functions with ipv6 segment routing*. In Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, pages 67–72, 2018.

[5] V.-H. Tran and O. Bonaventure. *Making the Linux TCP stack more extensible with eBPF*. In Proc. of the Netdev 0x13, Technical Conference on Linux Networking, 2019.

[6] A. Gulenko, M. Wallschläger, and O. Kao. *A practical implementation of in-band network telemetry in open vswitch*. In 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pages 1–4. IEEE, 2018.

[7] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke. *Tighter application-network interfacing to drive innovation in networked systems*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 53–57, 2021.

[8] W. Eddy. *Rfc 9293: Transmission control protocol (tcp)*, 2022.

[9] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *CUBIC for fast long-distance networks*. Technical report, 2018.

[10] S. Ha, I. Rhee, and L. Xu. *CUBIC: a new TCP-friendly high-speed TCP variant*. ACM SIGOPS operating systems review, 42(5):64–74, 2008.

[11] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. *Restructuring endpoint congestion control*.

In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pages 30–43, 2018.

3

Network- and Application-aware Adaptive Congestion Control Algorithm

In this chapter a network- and application-aware CC algorithm is designed by taking advantage of the connection between CC techniques and network context. The intermediate node is set up to process the application request (aided by APP-NET and INT) and assign capacity to each traffic flow in the network by utilizing the data programmability of the forwarding plane. The end device's CC algorithm along with INT parameters receives additional information about this allocation and adjusts the data transfer rate in response to it, taking into account the current network conditions and feedback. As a result, the algorithm's design takes into account both the needs of the application and shifting network conditions. Apart from offering a service free of congestion, the algorithm also attends to the requirements of the application and offers precise service differentiation.

⋆ ⋆ ⋆

This chapter is based on:
Network- and Application-aware Adaptive Congestion Control Algorithm

Published in Journal of Communications and Networks, June 2024.

⋆ ⋆ ⋆

3.1 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). Looking 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. 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, APP-NET integration, has enabled applications to specify their traffic and monitoring requirements to the network layer [1]. 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 INT. INT is capable of collecting node characteristics along with wireless link information in real-time which can be accessed by the application layer [2]. Previously in [3], INT and APP-NET have been used to modify the existing CUBIC CC algorithm for wireless networks. The obtained results indicate that utilizing INT information can greatly improve the overall performance of the CC algorithm. Therefore, in this study, the problem of congestion and differentiation of services is addressed by designing a rule-based network- and application-aware adaptive CC

[1] Deterministic Networking: https://datatracker.ietf.org/wg/detnet/about/
[2] Time-Sensitive Networking Task Group: https://1.ieee802.org/tsn/

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

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

The rest of the chapter is structured as follows, Section 3.2 discusses the related works in the area of transport protocols. Section 3.3 discusses the goals and Section 3.4 provides the network architecture overview. Section 3.5 describes the design and implementation of network-aware CC algorithm. The design and implementation of NACC is elaborated in Section 3.6. Section 3.7 presents the results and discussions of the performance evaluation of NACC. Section 3.8 concludes the chapter with insights on future works.

3.2 Related work

Over the years there has been several works to improve mechanisms of transport layer protocol. The Accurate-ECN [4] 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 [5] have designed a multi objective CC algorithm that adapts itself to different application requirements and corresponding optimal control policies. The CC 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 CC algorithm has been designed by the authors of [6]. The algorithm obtains an optimal $cwnd$ 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 [7] adjust the bitrate by predicting the Quality of Efficiency (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 [8]. 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 [9]. This approach does not yet consider the congestion and detailed QoS specific to each application. Also, authors in [10] 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 [11]. 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 CC 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 CC 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.

3.3 Goals

To achieve **congestion-free differentiation of services**, certain goals have been considered 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 packet losses due to channel conditions such as interference, mobility, obstacles, etc.. 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.

3.4 Network architecture and enablers

To realize the above mentioned goals, the network architecture that provides continuous monitoring data, application network interaction and a framework to reconfigure the CC algorithm is necessary. These three frameworks are indicated in the Figure 3.1 as (a), (b) and (c) and explained in this section.

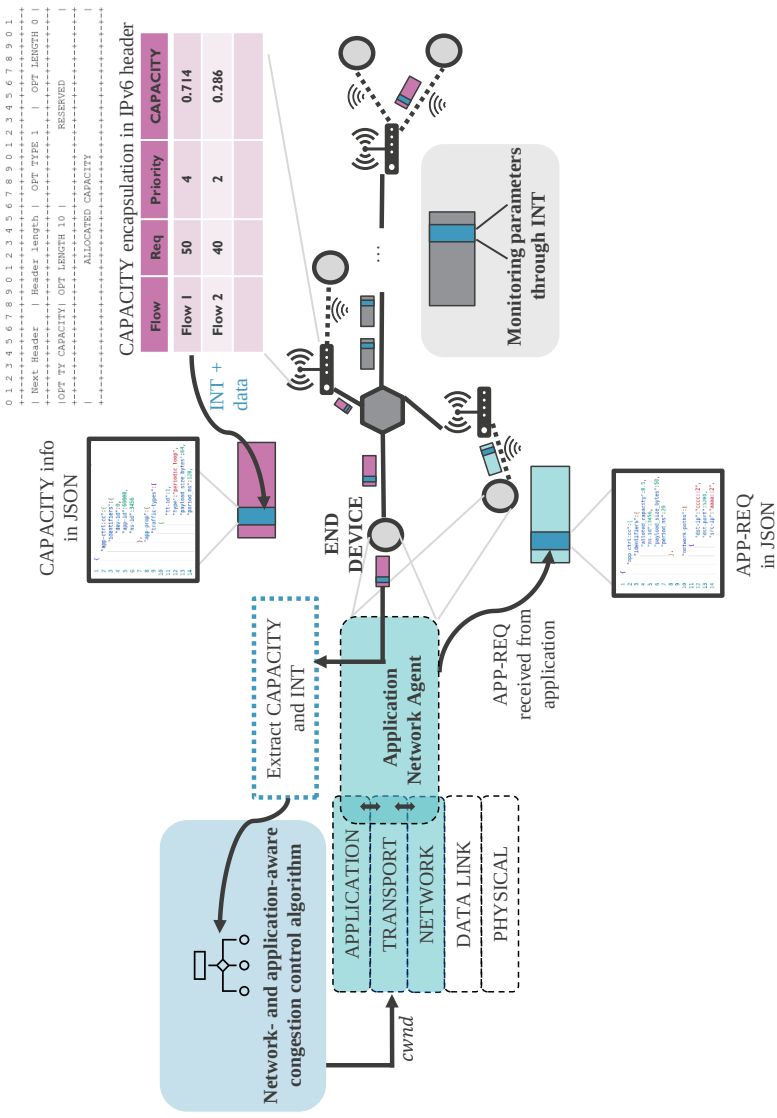


Figure 3.1: Network architecture overview

3.4.1 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 the INT which has been extended to wireless networks by designing new INT-enabled node architecture and logic to process INT-enabled packets for Wi-Fi-based networks [2] is used. 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, Figure 3.1). 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.

3.4.2 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 the APP-NET framework designed in [1], 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 (Figure 3.1) is used. 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, $priority$) 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 Synchronize (SYN) packet.

3.4.3 Reconfigurability of CC algorithms

In order to design NACC, CCP API is used [12] which provides a framework to reconfigure the CC algorithms. It allows us to write CC algorithms in the Python programming language and run them in any environment.

[3] P4 programming documentation: https://p4.org/p4-spec/docs/INT_v2_1.pdf

[4] Data Fields for In-situ Operations, Administration, and Maintenance: Data Types and Formats: https://tools.ietf.org/html/draft-ietf-ippm-ioam-data-04#section-4

Table 3.1: 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}	Data arrival rate	bits/second	INT
fc	No. of flows passing through the network node	no. of flows	INT
$loss_w$	Packet loss due to the quality of wireless medium	packets	INT
dr	Data rate of the link	Mbps	INT
rtt	Round Trip Time (RTT)	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
$cwnd_{prev}$	$cwnd$ 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

3.5 Network-awareness: Design and Implementation

3.5.1 Relevant monitoring parameters

Before designing the CC algorithm based on INT, it is important to revisit the relationship between different parameters obtained from INT and the $cwnd$ of TCP [3]. In existing CC 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 RTT of the packets. The existing TCP CC 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_data), flow count (fc), wireless packet loss ($loss\_w$), 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 CC algorithm 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.

While sending the SYN packet, the $cwnd$ is set to the same value as *ssthresh* in conventional CC 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 Equation 3.1, Equation 3.2 and Equation 3.3 respectively. The value of $cwnd_l$ is equal to the $ssthresh$. a_{up} is the upper bound and the data arrival rate at the intermediate node should not exceed above this value. 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), RTT, and 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. 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.

$$cwnd_l = 10 \tag{3.1}$$

$$cwnd_{up} = cwnd_l + \frac{(dr * 0.16) * rtt}{8 * MSS} \tag{3.2}$$

$$a_{up} = dr * 0.16 \tag{3.3}$$

Generally, CC algorithms have different phases such as slow start, congestion avoidance, fast recovery, and fast retransmit. As mentioned in Section 3.3, 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 ACK (or SACK) notification (on_ack) and the second

[5]Wi-Fi traffic distribution between control, management and data frames: https://www.itweb.co.za/content/o1Jr5qx96jDvKdWL

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 $cwnd$ ($cwnd_{prev}$) is reduced by the percentage of fc and a_{diff} (Equation 3.4). a_{diff} is the rate of increase in the load at the intermediate node. Once the $cwnd_{prev}$ is set based on fc, the algorithm will compare its value with various thresholds. Starting with $cwnd_l$, if the $cwnd_{prev}$ is lower than this threshold, then the $cwnd$ is aggressively increased until it exceeds $cwnd_l$ (Equation 3.6). Then the $cwnd_{prev}$ is compared with $cwnd_{up}$ and if $cwnd_{prev}$ is greater than $cwnd_{up}$, then the $cwnd$ is the same as the $cwnd_{up}$. If not, then the current a_{data} is compared with the a_{up}. If a_{data} 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, Equation 3.5). If not, then $cwnd$ is increased proportionally to the q, a_{data} and 1/rtt (Equation 3.6), 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.

$$cwnd_{prev} = cwnd * (1 - fc\% - a_{diff}\%) \tag{3.4}$$

$$cwnd = cwnd_{prev} * 0.9 \tag{3.5}$$

$$cwnd = cwnd_{prev} + \frac{q}{a_{data} * 8 * MSS} \tag{3.6}$$

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 $cwnd$ [3]. The conventional CC algorithm reduces the $cwnd$ by 30% on packet loss irrespective of the reason for the packet loss. The $cwnd$ was reduced 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 (Equation 3.7).

$$cwnd = cwnd_{prev} * 0.9 \tag{3.7}$$

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 Equation 3.1, Equation 3.2 and Equation 3.3 respectively.

The above explained algorithm is indicated as a flow chart/algorithm in the Figure 3.2.

3.5.2 Design of network-aware CC algorithm

The designed network-aware CC algorithm uses real-time monitoring information obtained from INT [2] The design was implemented in Python 3.7 programming language using CCP [12].

3.6 Application awareness in network-aware CC 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 CC algorithm of TCP tries to achieve fairness between the flows irrespective of their priority and traffic needs. The aim is to design an algorithm that provides fairness based on APP-REQ of the flows (cf., Section 3.3). This section elaborates on how application awareness is achieved in the network-aware CC algorithm.

3.6.1 APP-REQ processing in intermediate nodes: Data encapsulation and extraction

As explained in Section 3.4.2, 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 Equation 3.8) in JSON format along with the application identification as shown in the Listing 3.1. Equation 3.8 calculates the capacity, $capacity_r$ of flow r with $payload_r$, $period_r$ and priority $priority_r$ requirements and *n* flows in the node. This data structure is encapsulated in the IPv6 extension header (Figure 3.1) and the source node can extract the $capacity$ information from the INT packet.

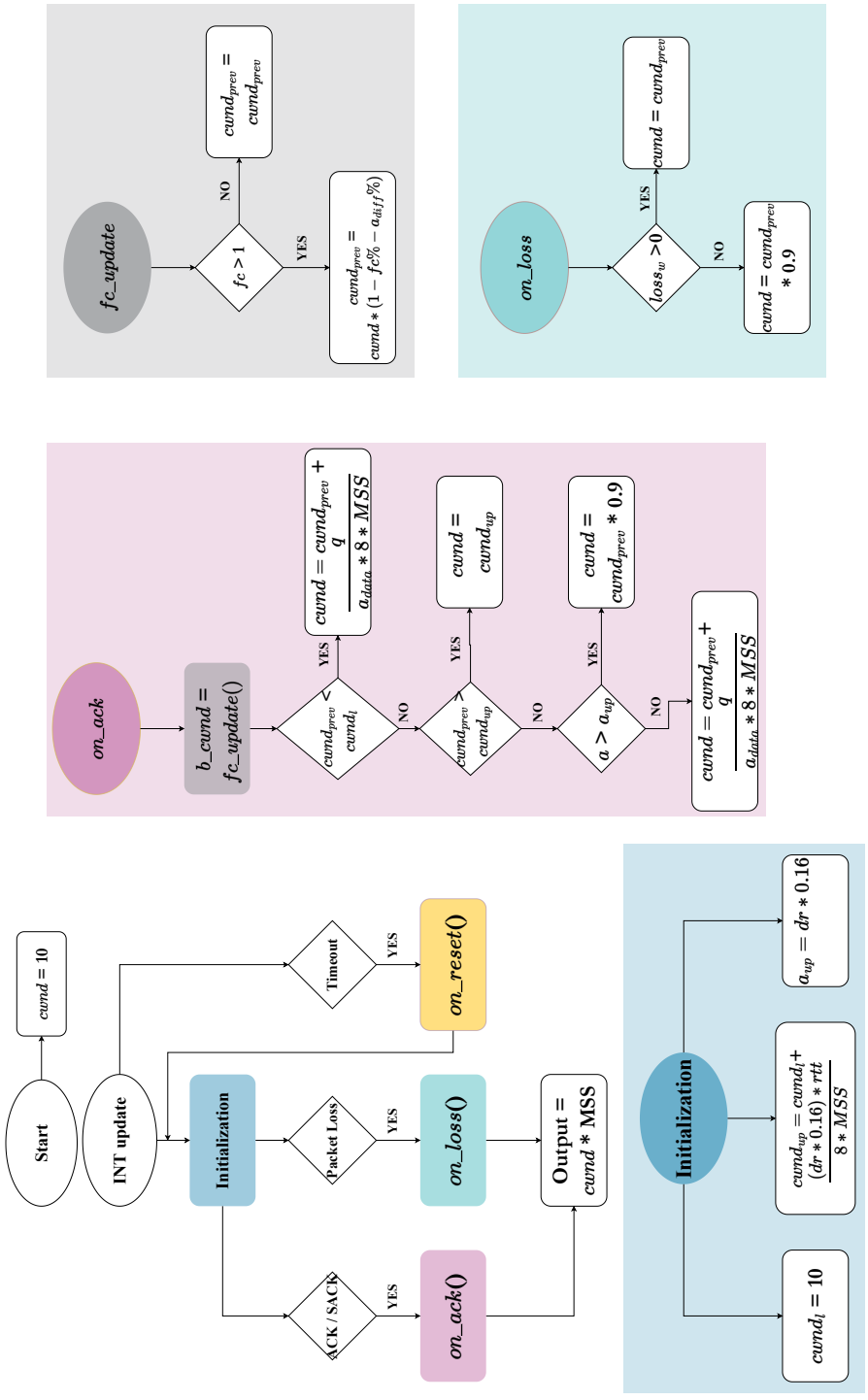


Figure 3.2: Flowchart of network-aware CC algorithm

$$capacity_r = \frac{priority_r * \left(\frac{payload_r}{period_r}\right)}{priority_1 * \left(\frac{payload_1}{period_1}\right) + ... + priority_n * \left(\frac{payload_n}{period_n}\right)} \tag{3.8}$$

```
{
    "app-ctrl:cc":{
        "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"
            }
        ]
    }
}
```

Listing 3.1: JSON data structure shared by AP

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 (Equation 3.8) 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.

The designed NACC (explained in Section 3.6.2) utilizes this updated capacity information to modify the data transfer rate and adapt to other flows in the network.

3.6.2 Making the network-aware CC algorithm application-aware

The network-aware CC 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. A few modifications are made to the network-aware CC algorithm mentioned in Section 3.5 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 $cwnd_{prev}$ 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 $cwnd_l$ (Equation 3.9). 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 $cwnd_{up}$ (Equation 3.10). 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 CC algorithm are directly proportional to $priority$ and $capacity$ respectively.

$$cwnd_l = 10 * priority_r \tag{3.9}$$

$$cwnd_{up} = cwnd_l + \frac{capacity_r * (dr * 0.16) * rtt}{8 * MSS} \tag{3.10}$$

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_{data} (Equation 3.11) and modifying the decrease rate of $cwnd$ on packet loss and making it inversely proportional to $priority$ (Equation 3.12).

These simple modifications to the network-aware CC algorithm make the algorithm application-aware. In this chapter, a NACC is designed with the option to switch to only a network-aware CC algorithm. Though the real-time network

parameters and APP-REQ of other flows in the network are utilized, there are still certain parameters that are not accessible to the transport layer and can result in packet loss. Hence the algorithm is designed 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 router framework[6]. The modifications in the CC algorithm are made in the previous implementation of the network-aware CC algorithm using CCP.

$$cwnd = cwnd_{prev} + \frac{priority_r * q}{a_{data} * 8 * MSS} \tag{3.11}$$

$$cwnd = cwnd_{prev} * (1 - 0.1 * (max_{priority} - priority_r)) \tag{3.12}$$

The above explained algorithm is indicated as a flow chart/algorithm in the Figure 3.3.

3.7 Results and discussion

3.7.1 Responsiveness of the NACC

As discussed in Section 3.5.1, key parameters such as q, a_{data}, 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 the impact of these parameters on the designed CC algorithm was tested. 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 (Figure 3.4a). All the devices use Ubuntu 18.04 as operating system, with kernel version 4.15.0-45-generic, TCP version as per the standard in [13] and IP version 6. APs are operating at 2.4 GHz frequency, i.e., using Wi-Fi IEEE 802.11g. 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_{data}, fc, and dr of the network.

With a sudden increase in a_{data} in Figure 3.5, the CC 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 AP suddenly decreases in Figure 3.6, the $cwnd$ also decreases. When the value of the dr at the AP decreases as shown in Figure 3.7, the $cwnd_{up}$ is reset according to this change and thus the designed algorithm also reduces the data transfer rate by lowering $cwnd$. Also when the number of flows in the network increases, the CC algorithm is notified by INT through fc, and thus the algorithm reduces the $cwnd$ as shown in Figure 3.8.

[6]Click modular router framework documentation: https://github.com/kohler/click
[7]Mininet-WiFi documentation: https://mn-wifi.readthedocs.io/en/latest/

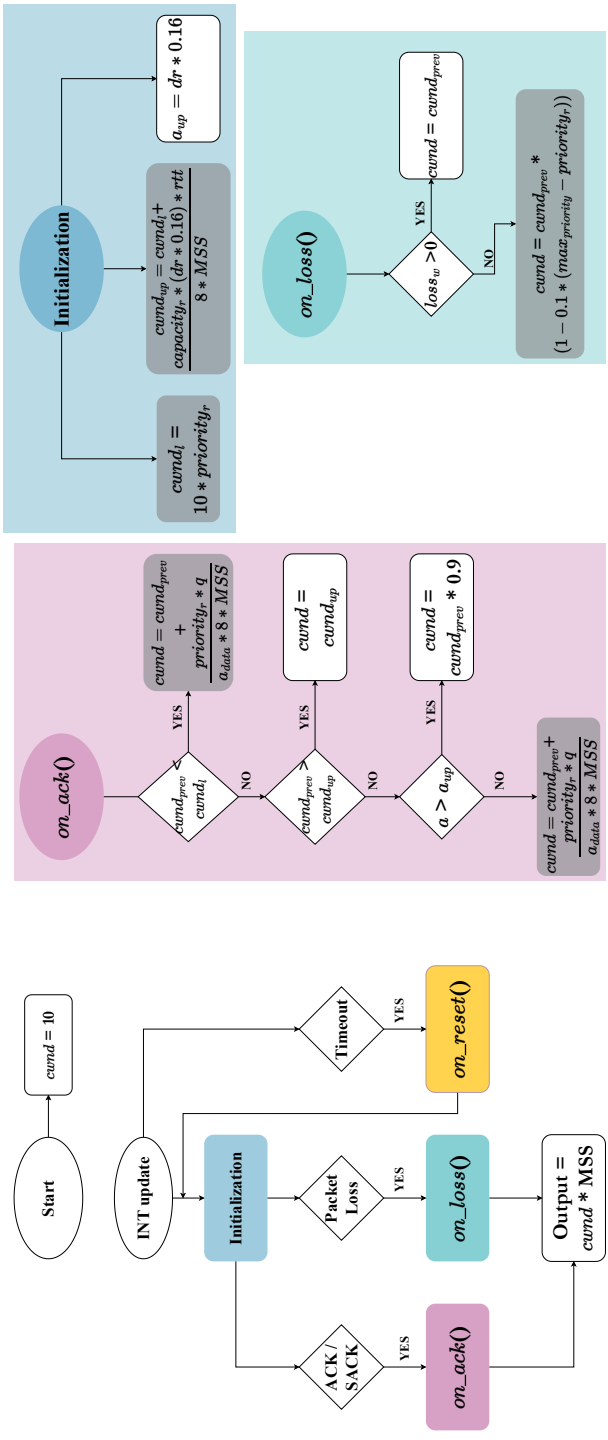
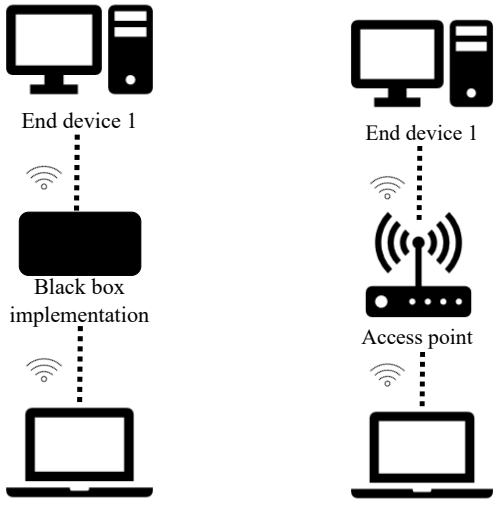


Figure 3.3: Flowchart of network- and application-aware CC algorithm



(a) Network emulated in Mininet-WiFi, with black box implementation.

(b) Network emulated in Mininet-WiFi, with an AP and two end devices.

Figure 3.4: Wireless network setup to test responsiveness of the algorithm and bench marking it against CUBIC algorithm.

Therefore it can be concluded that the designed CC algorithm is highly responsive to the changes in the network.

3.7.2 Benchmarking against the CUBIC CC algorithm

The authors in [14] and [15], have proven the CUBIC CC algorithm to be better than other CC 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 the designed NACC algorithm has been benchmarked against the CUBIC CC 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 AP as show in Figure 3.4b. All the devices use Ubuntu 18.04 as operating system, with kernel version 4.15.0-45-generic, TCP version as per the standard in [13] and IP version 6. AP is operating at 2.4 GHz frequency, i.e., using Wi-Fi IEEE 802.11g.

Figure 3.9 indicates that the throughput of the designed CC algorithm is better

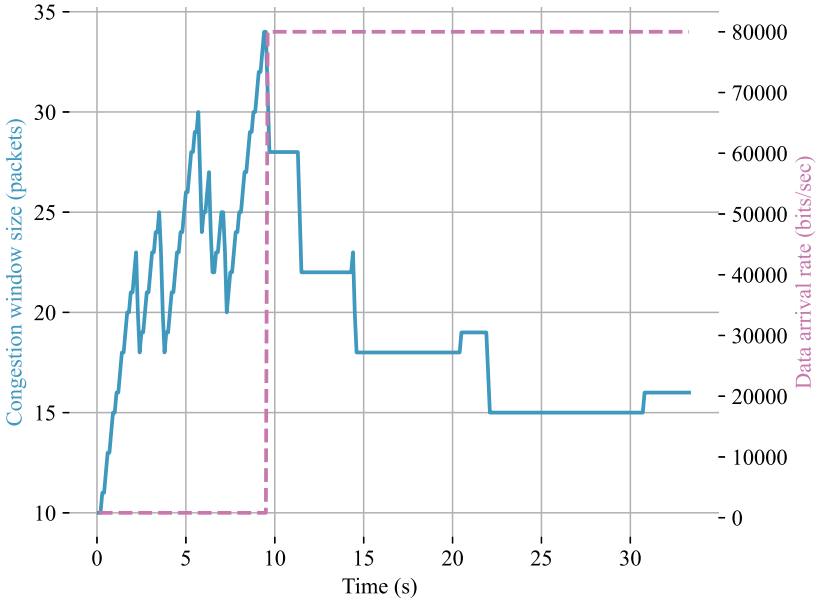


Figure 3.5: Responsiveness of cwnd of NACC to data arrival rate.

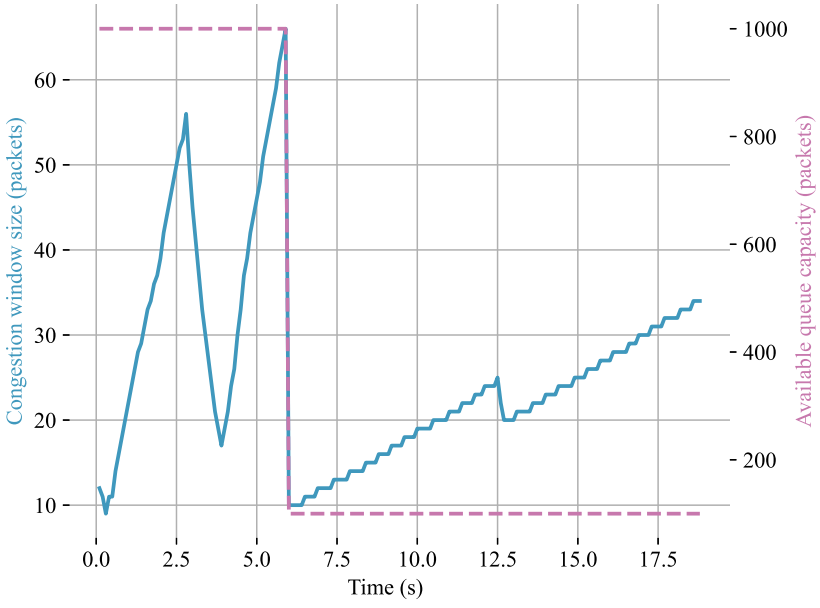


Figure 3.6: Responsiveness of cwnd of NACC to available queue capacity.

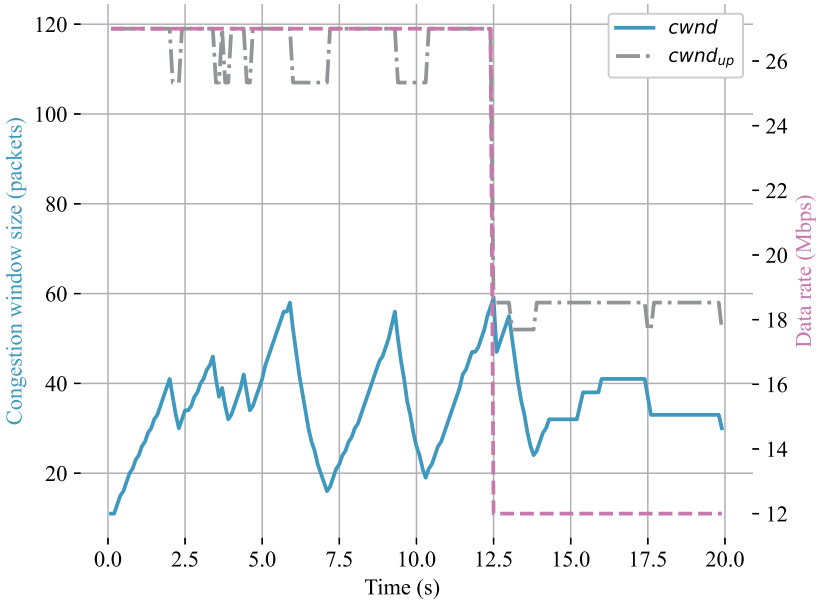


Figure 3.7: Responsiveness of cwnd of NACC to data rate of the wireless channel.

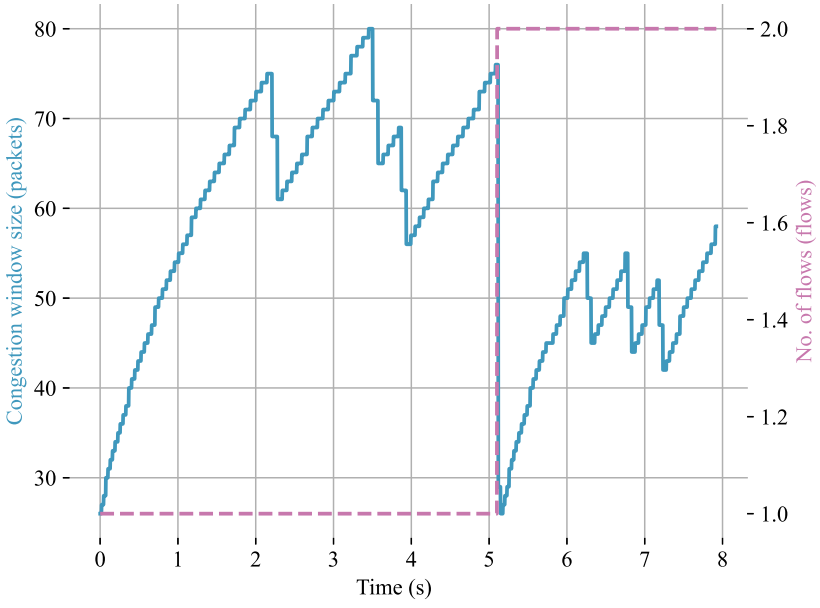


Figure 3.8: Responsiveness of cwnd of NACC to number of flows in the network.

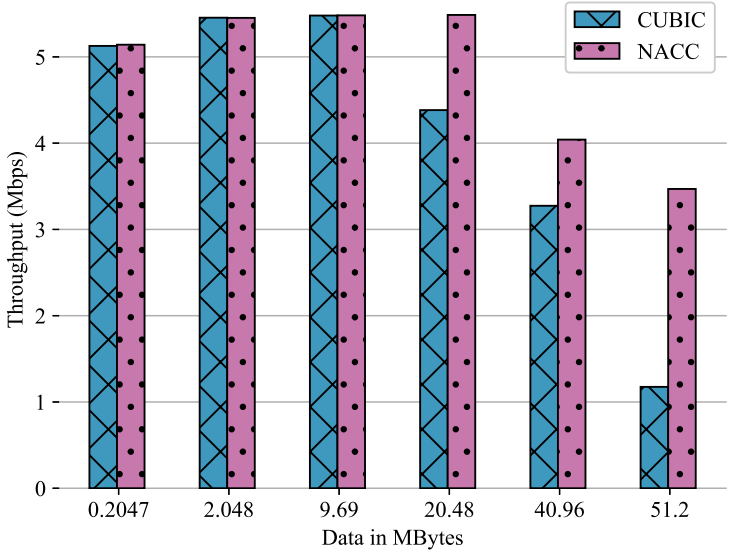


Figure 3.9: Throughput comparison of CUBIC and NACC.

than the CUBIC. Especially with the increase in the load (bytes sent), the throughput of the designed CC algorithm is almost three times better than the CUBIC. On starting the data transfer, the designed CC 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 CC algorithm reaches the maximum possible $cwnd$ for the given network before the CUBIC algorithm as indicated by the grey line in Figure 3.10. From the frequency distribution Figure 3.11, it can be seen that the $cwnd$ of CUBIC is highly variable whereas the $cwnd$ of the designed CC algorithm is more stable and tries to maintain the same value. The impact of this can be seen on the frequency distribution of a_{data} over 335 seconds in Figure 3.12.

3.7.3 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 CC algorithm was evaluated in a multi-flow wireless network in the IDLab Testbed[8] on ZOTAC and DSS wireless

[8]imec iLab.t testbeds' documentation: https://doc.ilabt.imec.be/ilabt/wilab/

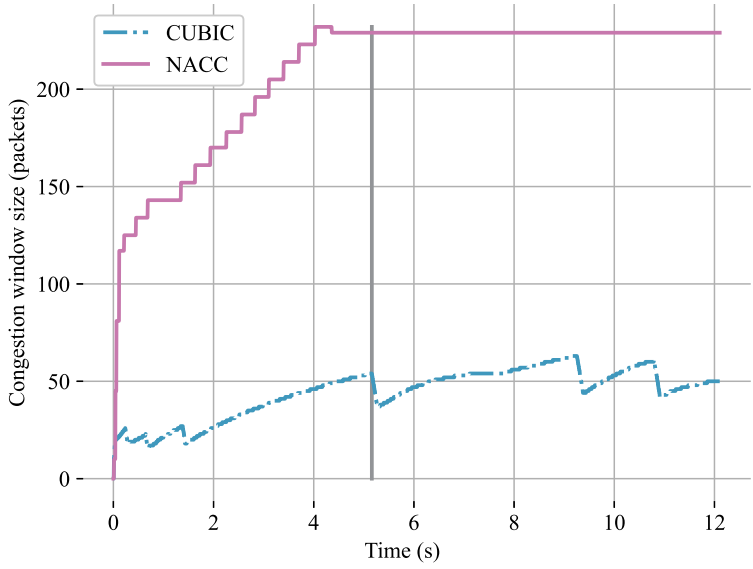


Figure 3.10: $cwnd$ over time of CUBIC and NACC.

nodes[9]. The testbed setup consisted of two APs (DSS nodes) connected by a wired connection and six end devices (ZOTAC nodes), a set of three connected to an AP wirelessly as shown in Figure 3.13. All the devices use Ubuntu 18.04 as the operating system, with kernel version 4.15.0-45-generic, TCP version as per the standard in [13] and IP version 6. APs are operating at a 2.4 GHz frequency, i.e., using Wi-Fi IEEE 802.11g.

To see the impact of priority levels on the $cwnd$ and the performance of the NACC algorithm, instead of real-time applications, socket programming was used to send the application data for a fixed amount of payload. This is useful to also compare the performance of NACC against the CUBIC algorithm. 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 3.1). 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 CC algorithm is plotted in Figure 3.14. 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 CC algorithm sets the $cwnd_l$ and $cwnd_{up}$ based on the

[9]w-iLab.2 hardware details: https://doc.ilabt.imec.be/ilabt/wilab/hardware.html#w-ilab-2-hardware

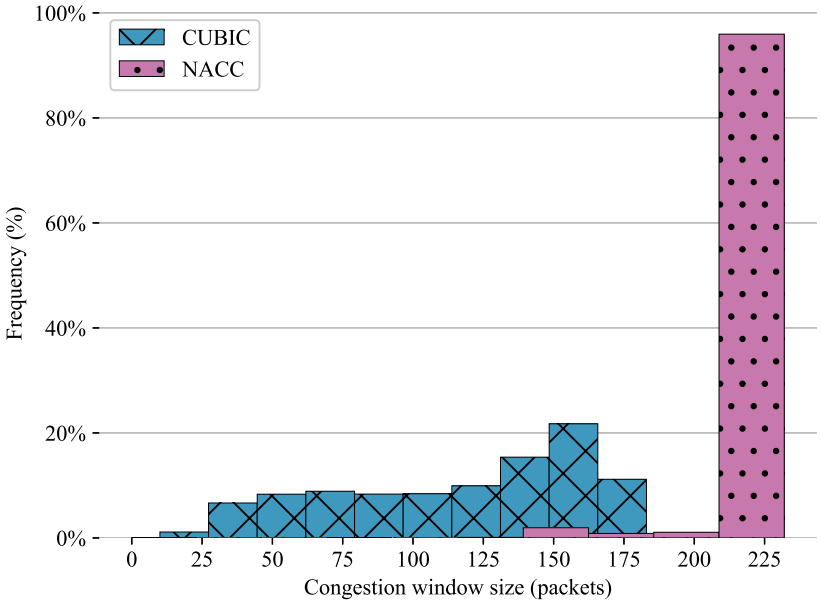


Figure 3.11: Frequency distribution of cwnd of CUBIC and NACC. Frequency % is the percentage of number of times the particular cwnd value is used.

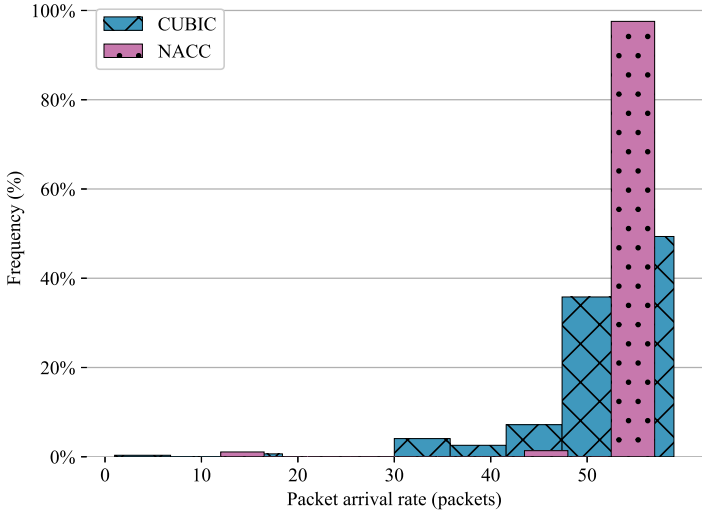


Figure 3.12: Frequency distribution of load at the intermediate node generated by CUBIC and NACC. Frequency % is the percentage of number of times the particular amount of packets arrived at the interface of intermediate network node.

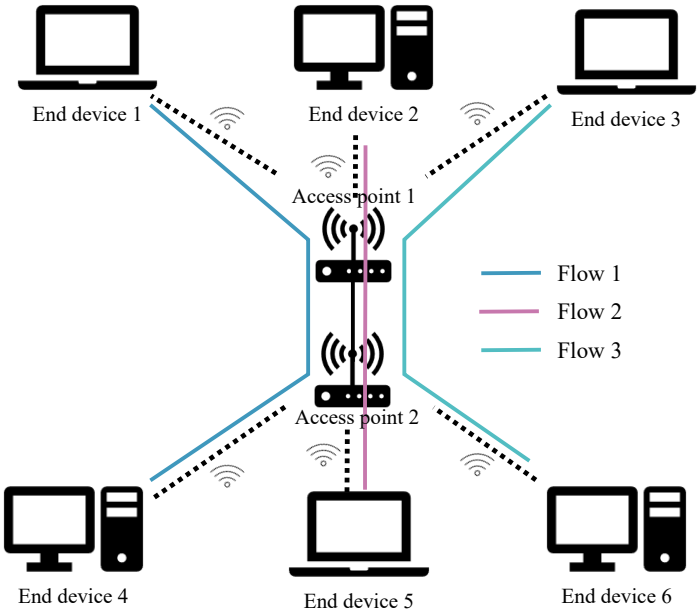


Figure 3.13: Multi-flow wireless network setup in IDLab Testbed.

APP-REQ, hence when the second flow starts, the AP 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 Figure 3.16. 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 Figure 3.15 indicate that on average the overall throughput of the NACC algorithm is twice the throughput of CUBIC. The difference can be seen 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 Figure 3.17. It can be observed that Flow 2 and Flow 1 have a priority difference of two, hence a larger gap in $cwnd$ as compared to Flow 1 and Flow 3 with a priority difference of only one.

3.8 Conclusion

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

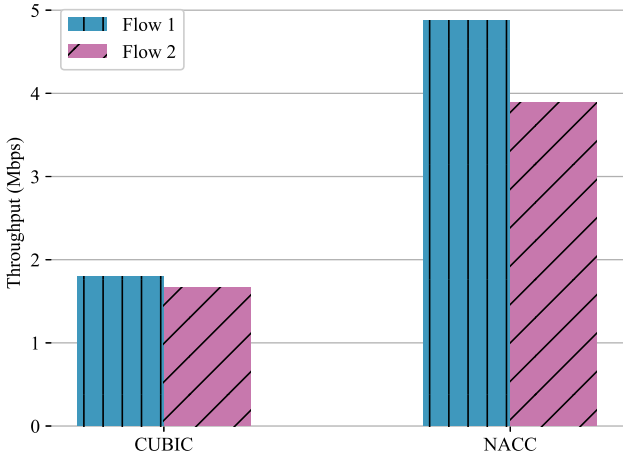


Figure 3.14: Throughput comparison of CUBIC and NACC for Flow 1 and Flow 2 with priorities 4 and 2 respectively.

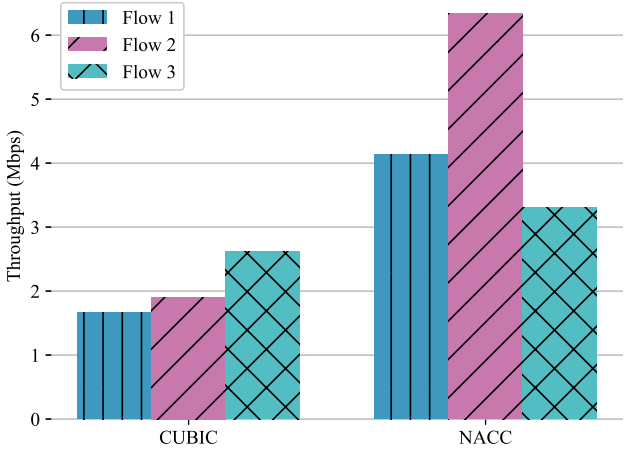


Figure 3.15: Throughput comparison of CUBIC and NACC for Flow 1, Flow 2, and Flow 3 with priorities 2, 4 and 1 respectively.

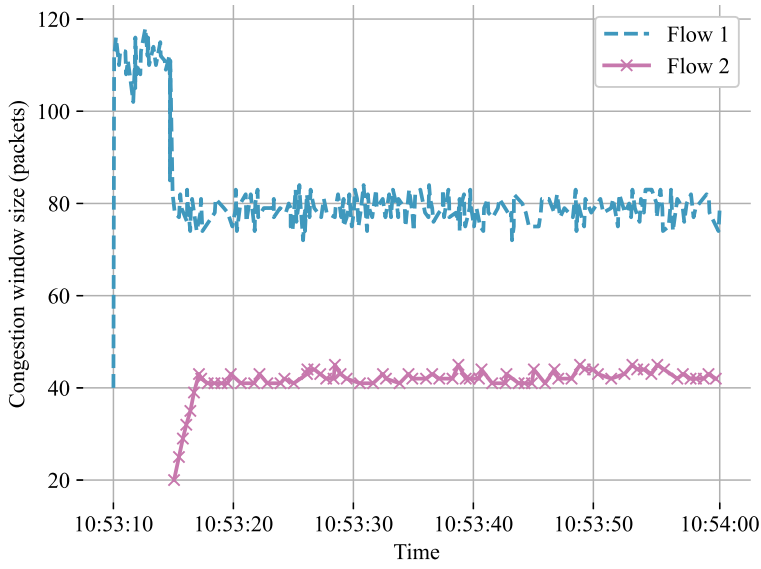


Figure 3.16: cwnd over time of NACC for Flow 1 and Flow 2.

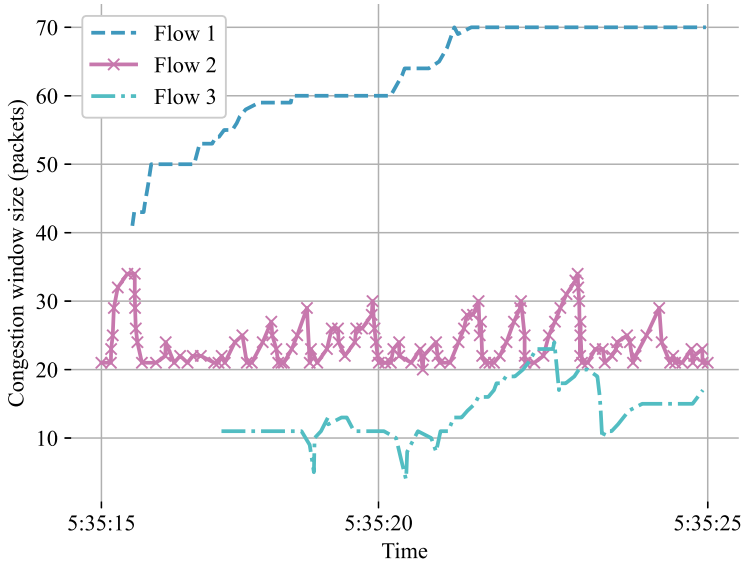


Figure 3.17: cwnd over time of NACC for Flow 1, Flow 2, and Flow 3.

aware of APP-REQ. In Chapter 2, the feasibility and benefits of utilizing these features were verified. In this chapter, a rule-based network- and application-aware adaptive CC algorithm is designed, 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 CC algorithm. The designed algorithm achieved three times more throughput than the CUBIC when 50 MB of application 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 CC 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.

References

[1] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke. *Tighter application-network interfacing to drive innovation in networked systems*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 53–57, 2021.

[2] 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 Transactions on Network and Service Management, 18(1):627–641, 2020.

[3] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke. *Adaptive transport layer protocols using in-band network telemetry and eBPF*. In 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pages 241–246. IEEE, 2021.

[4] J. Liu, S. Lu, and Q. Yang. *Accurate-ECN: An ECN enhancement with inband network telemetry*. In 2022 IEEE 47th Conference on Local Computer Networks (LCN), pages 375–378. IEEE, 2022.

[5] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin. *Multi-objective congestion control*. In Proceedings of the Seventeenth European Conference on Computer Systems, pages 218–235, 2022.

[6] L. P. Verma, V. K. Sharma, M. Kumar, and D. Kanellopoulos. *A novel delay-based adaptive congestion control TCP variant*. Computers and Electrical Engineering, 101:108076, 2022.

[7] T. Kimura, T. Kimura, A. Matsumoto, and K. Yamagishi. *Balancing quality of experience and traffic volume in adaptive bitrate streaming*. IEEE Access, 9:15530–15547, 2021.

[8] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. *The quic transport protocol: Design and internet-scale deployment*. In Proceedings of the conference of the ACM special interest group on data communication, pages 183–196, 2017.

[9] J. Zhang, L. Contreras, K. Gao, F. Cano, P. Cano, A. Escribano, and Y. R. Yang. *Sextant: Enabling automated network-aware application optimization in carrier networks*. In 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 586–593. IEEE, 2021.

[10] 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*. arXiv preprint arXiv:2112.04602, 2021.

[11] T. Krüger and D. Hausheer. *Towards an api for the path-aware internet*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 68–72, 2021.

[12] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. *Restructuring endpoint congestion control*. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pages 30–43, 2018.

[13] W. Eddy. *Rfc 9293: Transmission control protocol (tcp)*, 2022.

[14] S. Ha, I. Rhee, and L. Xu. *CUBIC: a new TCP-friendly high-speed TCP variant*. ACM SIGOPS operating systems review, 42(5):64–74, 2008.

[15] T. Kozu, Y. Akiyama, and S. Yamaguchi. *Improving RTT fairness on cubic TCP*. In 2013 First International Symposium on Computing and Networking, pages 162–167. IEEE, 2013.

[16] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *RFC 8312: CUBIC for Fast Long-Distance Networks*, 2018.

4

In-band Network Telemetry-based Congestion Control Algorithm for Industrial Wireless Networks

This chapter introduces the RACC algorithm, a reactive airtime feedback-based CC solution for wireless networks. The designed technique was developed to reactively adjust the application data transmission rate based on airtime allocation feedback from the AP on commercial off-the-shelf wireless devices. Based on the real-time network status, the suggested CC method tackles the problem of airtime unfairness and performance degradation in industrial wireless networks.

⋆ ⋆ ⋆

This chapter is based on:
In-band Network Telemetry-based Congestion Control Algorithm for Industrial Wireless Networks

Published in ETFA, September 2024.

⋆ ⋆ ⋆

and

⋆ ⋆ ⋆

Feedback-based Control loop Congestion Control Algorithm for Wireless Networks

Published in IEEE Access, July 2024.

⋆ ⋆ ⋆

4.1 Introduction

IEEE 802.11, known commercially as Wi-Fi, is one of the crucial tools for connectivity in smart factories. Industries are moving towards wider incorporation of Wi-Fi, and the protocols are expected to provide real-time, reliable, and quality data transfer services[1]. They must also address some of the existing wireless challenges like flexibility, interference, and reliability. Unlike the present Wi-Fi protocols being limited to a few applications, the future Wi-Fi protocols are expected to be flexible and cater to the needs of a wide variety of industrial applications.

A Wi-Fi network consists of an AP with multiple end devices connected to it. The physical data rate of the end device is determined by the channel quality which varies based on the distance of the end device from the AP, obstacles in the environment, interference from other end devices, etc. Hence in a Wi-Fi network, each end device can have a different physical data rate based on the channel conditions.

Since Wi-Fi uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), a pseudo-random fairness-based channel access method, in a mixed physical data rate network, the airtime consumed by the end device with a lower physical data rate is larger than that of an end device with higher physical data rate. This unfairness in airtime consumption slows down the end devices with higher physical data rates and degrades the performance of the entire network (elaborated in Section 4.3). Therefore, in a mixed physical data rate Wi-Fi network, the throughput of all the end devices irrespective of their physical data rates is degraded by the presence of an end device with very low physical data rate [1], [2], and [3].

The diversity of application requirements that utilize Wi-Fi networks in private-professional environments as well as the possibility of totally different channel conditions for different devices part of the same network requires differentiation in catering to such applications and end devices. Some industrial applications require high throughput, while other applications require higher reliability. To prioritize the demands of diverse applications, the Enhanced Distributed Channel Access (EDCA) mechanism is standardized [4], where traffic flows are distinguished in different QoS classes. However, the airtime unfairness and throughput degradation due to a multitude of physical data rates across end devices are still evident for the same QoS

[1]Industrial wireless technology special report: https://iebmedia.com/technology/industrial-5g-wireless/industrial-wireless-technology-2022-special-report/

class. Therefore, it is essential to address the problem of throughput degradation in mixed physical data rate Wi-Fi networks, so not all devices will suffer from it.

TCP is a widely used transport protocol as it provides ordered, reliable, and error-free data transfer utilizing its CC and flow control algorithms. TCP provides logical host-to-host communication service and thus has only an end-to-end view of the network. Therefore, the current CC algorithms employed by TCP operate based on the partial end-to-end information available from the ACK packets. Moreover, the existing CC algorithms have been designed to maximize the utilization of wired networks. However, for wireless networks, in addition to the physical data rate, CC algorithms also impact the airtime usage of individual devices (elaborated in Section 4.3).

Using concepts like APP-NET [5] and INT [6] for wireless networks, applications can communicate their service requirements to the network and simultaneously learn the real-time network state through continuous, low-overhead network monitoring. This results in an application-transport-network layer interaction system where the transport layer is aware of the application requirements and the real-time end-to-end network information. The eBPF has enabled the possibility of modifying the transport layer protocol including the design of the CC algorithm [7].

Utilizing these innovations, the issue of airtime unfairness leading to throughput degradation, and lack of CC algorithms suitable for industrial wireless networks is addressed by designing a reactive airtime feedback-based CC (RACC) algorithm. The designed algorithm immediately reacts to the airtime feedback from the AP by adapting the application data transfer rate, hence the name. The design was implemented in Commercial Off-The-Shelf (COTS) devices. The performance of the designed algorithm was validated in terms of throughput, airtime consumption, and efficiency and is proven to achieve airtime fairness among the end devices in a mixed physical data rate wireless network.

The rest of the chapter is structured as follows: Section 4.2 discusses the existing works in the field of airtime fairness; Section 4.3 gives background on unfairness in airtime usage in Wi-Fi networks; Section 4.4 discusses the challenges and problem statement addressed in the chapter; Section 4.5 explains the underlying frameworks and monitoring parameters necessary for the design; Section 4.6 explains the design and implementation of the RACC algorithm. Section 4.7 consists of results and observations. The conclusions are addressed in Section 4.8.

4.2 Related work

Existing works on achieving airtime fairness in wireless networks can be categorized into two approaches, either modifications to the Medium Access Control (MAC) layer or imposing scheduling policies in the AP. In this section, different research works that explore airtime fairness in wireless networks based on these two cate-

gories are elaborated.

4.2.1 Research works on MAC layer modifications for airtime fairness

In [8], the authors have designed an algorithm to calculate the weight of each node based on their spatial condition and the number of interfering nodes in multi-hop wireless networks. The computed weights are used to define the fragmentation threshold and buffer size and are added as an additional feature to the 802.11 MAC protocol. The authors have shown that adding this feature improves the overall performance in terms of individual throughput, end-to-end throughput, and fairness. The implementation requires too much computational power and the authors do not address the robustness of the design with changing network conditions. The authors of [9] propose an algorithm that derives the optimal uplink and downlink transmission probabilities i.e., adaptive backoff based on the estimation of backlog sizes. One of the drawbacks is that the system is implemented and tested in a simulator whereas modifying the contention window size in the existing network nodes is a challenge. In [10], the authors have designed an adaptive algorithm that modifies the contention window based on the analysis of mean packet arrival rates at end nodes to achieve airtime fairness thus maximizing the throughput. The work does not discuss the implementation in real-time wireless networks and is confined to the stability region of the arrival rate. The authors of [11] propose a differentiated reservation algorithm to reduce the collision among contending nodes and overuse of shared channels by low bitrate nodes. Once the device accesses the channel, the designed algorithm assigns a deterministic value to the backoff counter and allows certain nodes to send multiple packets in one transmission. The authors also propose a Group-based Differentiated Reservation algorithm for high-density scenarios.

4.2.2 Research works enforcing scheduling policies in AP for airtime fairness

In [12], the authors have implemented and validated responsible airtime fairness through a fair scheduler in AP. The novelty of responsible airtime fairness lies in the fact that it achieves time-based fairness by including both data transmission time and the overhead of TCP ACK segments in TCP traffic and calculating the fairness index based on the throughput measurement. The authors do not consider the congestion in AP and the implementation is not tested in real-time wireless network environments. The authors of [13] have implemented a time-slotted scheduling approach to achieve fairness in content dissemination in mobile social networks. The authors achieve fair airtime allocation based on GO-coordinated dissemination as a generalized Nash bargaining game, where the GO-coordinated dissemination

schedules the data to be sent in the application layer. The algorithm is intended for Wi-Fi-direct only and its extension to Wi-Fi is not considered. The authors of [14] aim to achieve airtime fairness using the traffic control and queuing discipline features of the Linux kernel in AP. Here, the AP collects the link capacities of the nodes, calculates the weight and quota for each node, and sets the queuing discipline based on this quota. The implementation, however, lacks the complete utilization of available bandwidth and does not consider the congestion in AP. In [15], the authors have designed a new integrated queuing system in the Linux kernel to achieve low latency in Wi-Fi networks. Utilizing this, the authors have designed a deficit-based scheduler in AP to achieve airtime fairness. The new queuing system that bypasses the queuing discipline of Linux adds additional packet processing overhead and is less flexible. In [16], the authors have designed a REACT architecture where the end nodes bid their airtime requirements in an ad-hoc setting. The design has been extended to support differentiated airtime service as per the end node's requirement. The authors of [17] have analyzed the airtime distribution in Wi-Fi networks and tested the implications of Airtime Fairness technology implemented in AP equipment available on the market. The technology is based on Time Division Multiple Access, where each device gets equal or specific period of time for data transmission in cyclical order. The authors validate that the Airtime Fairness Technology works efficiently even in the presence of 20 or more users, networks with reduced coverage radius, and HotSpot networks. All the above research works, their novelties, and drawbacks are summarised in Table 4.1.

The modifications in the MAC layer generally involve setting the contention window size or buffer size based on various network parameters. In the case of scheduling policies, the APs allot specific transmitting time to each node based on different network parameters as well. Except for a few numerable works, most of the implementations are validated in a simulated environment. Existing research works also fail to consider the possible network congestion in AP, the robustness of the algorithm to changing network conditions, and their impact on a real-time wireless network. Since wireless network conditions are unpredictable, we take into account the real-time network conditions by utilizing INT in our design, resulting in a robust algorithm. Additionally, modification to commercially available network devices itself is a challenge, we specifically emphasize the fact of implementation and testing in real-time wireless networks and consider it as one of the novel features of this design.

4.2.3 Research works on adaptive CC algorithms

Unlike UDP, in a TCP connection, the receiver sends an ACK as it receives the data from the sender. Additionally, TCP detects packet loss based on timeout or 3

Table 4.1: Summary of research works in the area of achieving airtime fairness in wireless networks

MAC layer modifications

Reference	Novelty	Drawbacks
[8]	Additional feature to 802.11MAC: fragmentation threshold and buffer size	Too much computational power and do not address the robustness
[9]	Adaptive backoff based on the estimation of backlog sizes	Modifying the contention window size in the commercially available wireless devices itself is a challenge
[10]	Modifying the contention window size based on mean packet arrival rates	The implementation is confined to the stability region of arrival rate
[11]	Differentiated reservation algorithm: deterministic backoff counter value and allows the nodes to send multiple packets in one transmission	Increased collision with increase in number of nodes

Scheduling policies in AP

Reference	Novelty	Drawbacks
[12]	Fair scheduler: time-based fairness based on throughput measurement	Congestion in AP is not considered in the design
[13]	Fair airtime allocation based on GO-coordinated dissemination	Implemented for Wi-Fi direct and is not extended to Wi-Fi
[14]	Traffic control and queuing discipline features of Linux kernel in AP to calculate the weight and airtime quota of each node	Does not completely utilize the available network bandwidth and does not consider the congestion in AP
[15]	Deficit-based scheduler in AP based on new integrated queuing system in Linux	The new queuing system adds additional packet processing overhead and is less flexible
[16]	REACT architecture where ends nodes bid their airtime in ad-hoc network	Not extended for infrastructure Wi-Fi networks
[17]	Time Division Multiple Access in specific AP equipment available in the market	The feature is not available in all the APs and does not support roaming clients

duplicate ACK events and retransmits the lost data to the receiver, thus achieving reliable and error-free data transfer. Among other mechanisms used by TCP, the CC algorithm decides the amount of data to be sent to the receiver at each instance of time by calculating the $cwnd$. Moreover, the CC algorithms are designed for high bandwidth wired data links to maximize the usage of the link while at the same time achieving fairness and avoiding congestive collapse. Once the $cwnd$ is decided, the segment is converted to a packet and then an IEEE 802.11 frame for wireless transmission. Once the frame is ready the sender contends for the channel access and transmits the frame when the channel is idle. Therefore, along with the physical data rate, the $cwnd$ set by the CC algorithm is also responsible for airtime consumed.

In [18], the authors have tried combining two traffic shaping methods, the Hierarchical Token Bucket shaping method (arbitrary number of token buckets are arranged in hierarchy) and the Receive Window Tuning Method (configure TCP connections by tuning receive window buffer size) with four different CC algorithms NewReno, Vegas, Illinois, and CUBIC, to improve the quality of experience of HTTP adaptive streaming. The authors have validated that the combination of the Receive Window Tuning Method and Illinois CC algorithm offers the best quality of experience for this application scenario. The authors of [19] have designed an adaptive CC method to avoid congestion in an IoT network with CoAP as an application layer protocol. The designed solution operates based on traffic priority assignment, adaptive retransmission timeout, and adaptive backoff timer. In [20], the authors have designed an adaptive CC to reduce the energy consumption in the Internet of Vehicles. The designed algorithm operates based on the two feedback bits indicating the status of buffer space and link utilization in the ACK packet, appended by the receiver. The above-mentioned adaptive CC algorithms are specific to particular use cases like HTTP adaptive streaming, Internet of Vehicles, and CoAP-based IoT networks. The algorithms rely on partial feedback from the network and are unaware of other devices and their airtime utilization.

4.3 Background on Unfairness in airtime usage in Wi-Fi networks

In a wireless network, unlike a wired network with fixed data rates, each end device connecting to an AP has a different physical data rate depending on the channel quality. For example, the latest Wi-Fi standard 802.11ax offers different physical rates ranging from 8.6 Mbps to 9.6 Gbps, i.e., the end device can choose a suitable modulation and coding scheme based on the channel quality. A few major factors that influence the channel quality in a Wi-Fi network are the signal strength and interference from other end devices. Hence, in a Wi-Fi network with multiple end

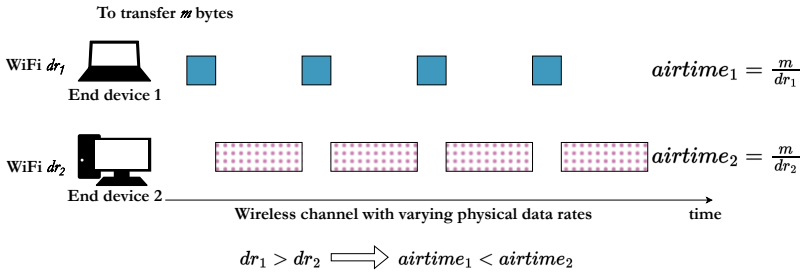


Figure 4.1: Difference in airtime in a wireless network with varying physical data rates

devices, each device would have a different physical data rate at each instance of time.

In a wired Ethernet network, the Carrier Sense Multiple Access with Collison Detection (CSMA/CD) technique is used to ensure pseudo-random fairness in media access among multiple end devices. This would mean that a wired link of *x* Mbps capacity would be divided equally among *n* end devices, resulting in a throughput of *x/n* Mbps. In the case of a Wi-Fi network, though CSMA/CA [21] ensures pseudo-random fairness in channel access, because of the changing wireless channel conditions, the estimation of throughput is not as simple as in a wired network. The end device with the poor wireless channel quality experiences a lower physical data rate and hence requires more airtime to transfer the same amount of data as compared to the end device with better channel quality and higher physical data rate. Therefore, to send *m* byte of data, the lower physical data rate device would consume more airtime than the higher physical rate device as shown in Figure 4.1. Airtime unfairness is an evident problem of Wi-Fi networks that is still to be addressed.

The side effect of airtime unfairness is the significant degradation in the performance of the end devices with higher physical data rates. This has been experimentally validated by the authors of [2] and [3]. They have proven that the throughput of end devices with higher physical data rates is significantly degraded in the presence of end devices with lower physical data rates.

Transmit Opportunity (TXOP) is a MAC feature in 802.11 that provides channel-free access to high-priority data for a certain period. TXOP limit is supported only for access categories, Voice and Video. The TXOP limit cannot be dynamically changed for each access category on the fly. Moreover, the TXOP feature is applied to an entire access category, hence it is applied to all the flows that pass by the dedicated queue for that access category. This restricts the application of different service treatments to different flows. If all the devices enable the TXOP feature, then the end device with a lower physical data rate would still slow down the end device with a higher physical data rate, hence the problem of airtime unfairness

persists between the flows belonging to the same access category.

4.4 Problem statement

This chapter aims to solve the issue of airtime unfairness in industrial wireless networks by designing a CC algorithm that adapts the application data transfer rates by reacting to the feedback from the network and AP. The performance of the designed algorithm is validated by evaluating the improvement in the throughput of a high physical data rate device when the RACC algorithm is used. The difference in the airtime consumption of a high physical data rate device when the RACC algorithm is used, as compared to the CUBIC CC algorithm, is also verified.

4.5 Underlying frameworks and Monitoring parameters

This section explains the underlying frameworks that facilitate the design of the RACC algorithm. Then discuss the essential monitoring parameters that are obtained as feedback from AP through INT.

4.5.1 Underlying frameworks

The design of the reactive airtime feedback-based CC algorithm is based on three concepts that have been recently presented in literature. The Figure 4.2 indicates the workflow of implemented design along with underlying frameworks.

This subsection gives a short explanation of each of the such enablers.

1. For the applications to react to the changes in the network it is essential to continuously receive real-time feedback from the network. For this, the algorithm is built upon the idea proposed by the authors of [6] regarding a multi-plane design capable of performing real-time monitoring. The wireless INT is a low overhead, real-time monitoring that enhances data packets with monitoring data on the fly. Monitoring data include end-to-end information and per-hop information such as physical data rate, number of flows in the path, RSSI, etc.

2. In the current OSI model, application and transport layers only have an end-to-end view of the communication and are uninformed of the underlying network. The APP-NET API designed by the authors of [5] has facilitated the interaction between application, transport, and network layers, resulting in application-transport-network layer interaction. Hence, this framework is used to communicate the application requirements from the application to

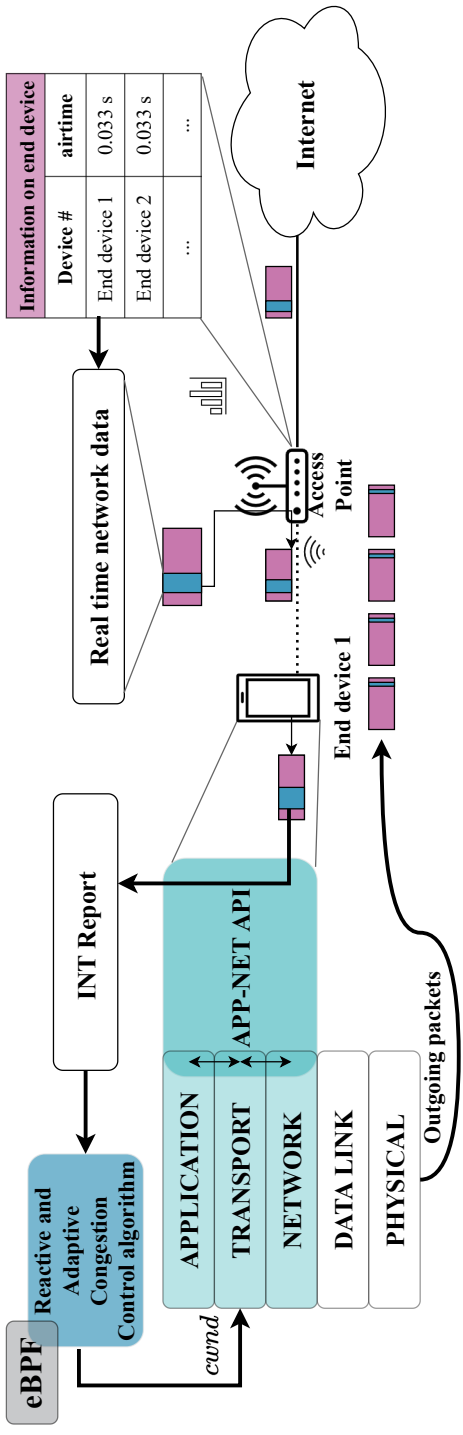


Figure 4.2: The block diagram of implemented wireless system with underlying frameworks.

the network layer as well as the network layer information to the application and transport layers of OSI model.

3. The CCP API, designed by researchers at MIT, is a framework based on eBPF, that allows the reconfiguration of CC algorithms [7]. This framework is used to implement the reactive airtime feedback-based CC algorithm.

4.5.2 Monitoring parameters

The key network parameters that directly affect the airtime consumption of the end devices are physical data rate (dr, unit: Mbps) and the number of end devices in the wireless network (dc, unit: number of end devices). The packet loss in an industrial wireless network is not only caused by network congestion but can be due to wireless conditions like interference. Hence, to differentiate the packet losses due to wireless conditions, the per-hop counter value (terminology from [6]) is also an essential parameter.

Since the AP has an overall view of the connected end devices, the designed algorithm depends on the feedback from the AP. To achieve airtime fairness, the AP maintains a table of airtime allocated to each end device, by calculating the $airtime_i$ (unit: second/device) of each end device using Equation 4.1:

$$airtime_i = \frac{ts}{dc} \tag{4.1}$$

where ts (unit: second) is fixed by the AP and gives the period over which the AP calculates the airtime allocation per end device, and dc indicates the number of end devices in the network seen by the AP. The $airtime_i$ indicates the amount of airtime that can be utilized by the device i. With the increase in the number of end devices, the airtime allocated to each device is decreased, therefore the $airtime\_i$ is inversely proportional to and dependent on the number of end devices. Since the aim is to achieve airtime fairness, the equation does not have any weighted component, and the given airtime is equally distributed among different flows.

Therefore the monitoring parameters that are sent as INT feedback are dr, $loss_w$, and $airtime_i$.

4.6 Reactive airtime feedback-based CC algorithm

This section describes the design and implementation of the reactive airtime feedback-based CC algorithm.

4.6.1 Design of RACC algorithm

This chapter aims to achieve airtime fairness among the end devices by adapting the application data transfer to the network and AP feedback. dr, $loss_w$, and $airtime_i$

are the monitoring parameters received as feedback from AP by the end device. For a given dr and allotted $airtime_i$, the number of packets that can be sent for each ts, i.e., the $cwnd$ (unit: packet) is given by the Equation 4.2:

$$cwnd = \lfloor \frac{(dr * airtime_i)}{MSS * 8} \rfloor \tag{4.2}$$

The multiplication of dr and $airtime_i$ gives the amount of payload, and dividing it by MSS gives the number of packets that can transferred for the given ts. The obtained $cwnd$ equation is dependent on dr and $airtime_i$ feedback from AP. Since the $cwnd$ is directly proportional to the dr, the high physical data rate device sends more packets during each data transfer as compared to the low physical data rate device. The increase in the number of end devices in the network, decreases the $airtime_i$, thus decreasing the number of packets sent at each data transfer by the device. Thus the designed algorithm aims to achieve airtime fairness by making the application data transfer rate dependent on the physical data rate and the changing network conditions.

4.6.2 Implementation of RACC algorithm

The designed RACC algorithm is represented graphically in Algorithm 2. On starting a new TCP connection, the RACC algorithm first enters the $Initialization$ phase where it receives the value of total $payload$ length (unit: byte) to be sent, via APP-REQ interface. It sets the value of $init\_cwnd$ (unit: packet), i.e., the $cwnd$ of the first data transfer, and β, where β is a $cwnd$ decrease factor after packet loss event. Once the initialized values are set the $reset\_function()$ is called as shown in Algorithm 2. The value of $init\_cwnd$ is set to 1, and the device sends an SYN packet, where the $cwnd$ is set to $init\_cwnd$.

On receiving this SYN packet, the AP registers the device and increases the dc. The AP then calculates the $airtime_i$ of the current device using Equation 4.1. For experimental purposes, the ts is fixed to 0.066 s (explained in Section 4.7.1). The AP reports the monitoring parameters in Section 4.5.2 as INT header. This information is received by the device as feedback with the SYN-ACK packet. This $airtime_i$ feedback is further sent at every ts, hence updating the device with ongoing changes in the network, for example beginning of a new connection from an end device.

If the connection is successful, the end device receives a SYN-ACK packet. The RACC algorithm then enters the on_report function through which it gets the information on $airtime_i$ allotted and other wireless network parameters. The algorithm also receives feedback about the end-to-end parameters of the ongoing connection such as $bytes_{ACK}$ (total bytes acknowledged, unit: byte), $bytes_{SACK}$ (total misordered bytes sent, unit: byte), $loss_p$ (total number of packets lost, unit: packet)

Algorithm 2 The designed RACC algorithm

Initialization:
Value updated from APP_REQ: $payload$, $\beta \leftarrow 0.9$,
$init\_cwnd \leftarrow 1$, $cwnd \leftarrow init\_cwnd$,
$cwnd_{out} \leftarrow MSS * cwnd$, $reset\_function()$

on_report**:**
Values updated from INT feedback:
$loss_w$, $airtime_i$, dr
Values reported by eBPF:
$bytes_{ACK}$, $bytes_{SACK}$,
$loss_p$, $flow\_timeout$, MSS
if $bytes_{ACK}$ or $bytes_{SACK}$ **then**:
    $on\_ack()$
else if $loss_p$ **then**:
    $on\_loss()$
else if $flow\_timeout$ **then**:
    $on\_timeout()$
end if
$on\_ack()$:
if $airtime \neq 0$ **then**:

$$cwnd \leftarrow \lfloor \frac{(dr * airtime_i)}{MSS * 8} \rfloor$$

else:
    $cwnd \leftarrow init\_cwnd$
end if:
$cwnd_{out} \leftarrow MSS * cwnd$

$on\_loss()$:
if $loss_w$ **then**:
    $cwnd \leftarrow cwnd$
else:
    $cwnd \leftarrow cwnd * \beta$
end if
$cwnd_{out} \leftarrow MSS * max(cwnd, init\_cwnd)$

$flow\_timeout()$:$reset\_function()$

$reset\_function()$:
$loss_w \leftarrow 0$, $airtime_i \leftarrow 0$, $dr \leftarrow 0$,
$bytes_{ACK} \leftarrow 0$, $bytes_{SACK} \leftarrow 0$, $loss_p \leftarrow 0$

and $flow\_timeout$ (if the TCP connection of the flow itself was lost due to TCP timeout, Boolean variable) from Linux kernel using eBPF.

If the device receives an ACK packet, the RACC algorithm enters the $on\_ack()$ function. The number of packets that can be sent after each ACK packet is calculated using Equation 4.2. If there is no $airtime_i$ feedback, the $cwnd$ value is set to $init\_cwnd$ to sustain the TCP connection. The same function is executed on receiving every ACK packet.

If the RACC algorithm receives a notification of $loss_p$, the algorithm enters the $on\_loss()$ function. It first checks if the packet loss is due to the wireless medium ($loss_w$, unit: packet), if so, it keeps the $cwnd$ constant. If the packet loss is due to the congestion in the intermediate nodes, then the $cwnd$ value is multiplied by β, thus decreasing the value by 10%. The conventional CUBIC CC algorithm uses the β value of 0.7 [22]. The performance of the RACC algorithm was tested for different β values of 0.7, 0.8, 0.9, and 0.95 in a congested network. The performance of the algorithm was proven to be better for the β value of 0.9. If the RACC algorithm receives the notification of connection timeout, it enters the $reset\_function()$. The $reset\_function()$ resets the values of $loss_w$, $airtime_i$, dr, $bytes_{ACK}$, $bytes_{SACK}$, and $loss_p$ to zero.

The INT monitoring and airtime feedback in AP are implemented in the Click modular router framework[2]. The designed RACC algorithm was implemented in Python 3.8 programming language, using the CCP framework [7].

4.7 Performance of the designed RACC algorithm

This section explains the test setup used to evaluate the performance of RACC algorithm. It then elaborates on the performance of the RACC algorithm in terms of throughput improvement and percentage airtime difference as compared to the CUBIC algorithm.

4.7.1 Test setup

The RACC algorithm aims to mitigate the airtime unfairness in wireless networks. To validate the performance of the algorithm, problem statements have been formulated in terms of airtime consumption and throughput as stated in Section 4.4. The experiments were conducted in w-iLab.2 testbed[3] on ZOTAC and DSS wireless nodes[4]. The experimental setup consisted of two APs (DSS nodes) connected by a wired connection and six end devices (ZOTAC nodes), with three end devices connected to each AP. All the devices use Ubuntu 18.04 as operating system, with

[2]Click modular router framework documentation: https://github.com/kohler/click

[3]imec iLab.t testbeds' documentation: https://doc.ilabt.imec.be/ilabt/wilab/

[4]w-iLab.2 hardware details: https://doc.ilabt.imec.be/ilabt/wilab/hardware.html#w-ilab-2-hardware

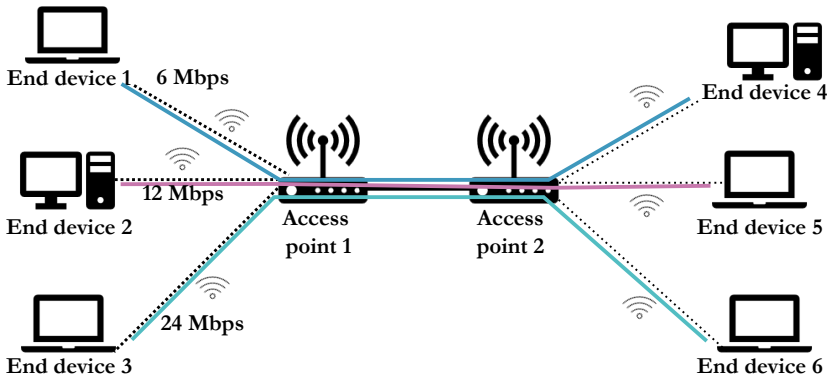


Figure 4.3: Wireless network setup

kernel version 4.15.0-45-generic, TCP version as per the standard in [23] and IP version 6. The APs are configured for 802.11n Wi-Fi standard and the aggregation is disabled. Both the APs operate in different wireless channels and are far enough to avoid interference. To clearly see the impact of the designed algorithm, each end device is restricted to one flow (indicated in different colors in Figure 4.3). To emulate different physical data rates in a network, the physical data rate of end devices 1, 2, and 3 are set to 6 Mbps, 12 Mbps, and 24 Mbps respectively, using the `iwconfig` command (only a few physical data rate values can be fixed using the `iwconfig` command and these are chosen for testing). The end devices 4, 5 and 6 connected to AP2 serve only as data receivers whose physical data rates are not fixed. The ts fixed for $airtime_i$ allocation should be large enough to include the RTT (unit: second) of all the end devices using the wireless link. Therefore, the ping tests were run in all the end devices and the ts in AP was set to the value of average RTT, 0.066 s, as shown in Table 4.2.

Table 4.2: Ping results of end devices

End device	**min (s)**	**avg (s)**	**max (s)**
End device 1	0.019491	0.067910	0.162247
End device 2	0.015527	0.066231	0.193776
End device 3	0.014046	0.063360	0.111578
Average (chosen timeslot)	-	0.066	-

As it is essential to see the impact of the designed algorithm on the consumption of airtime, instead of real-time applications, a fixed payload of 5 MB was sent from

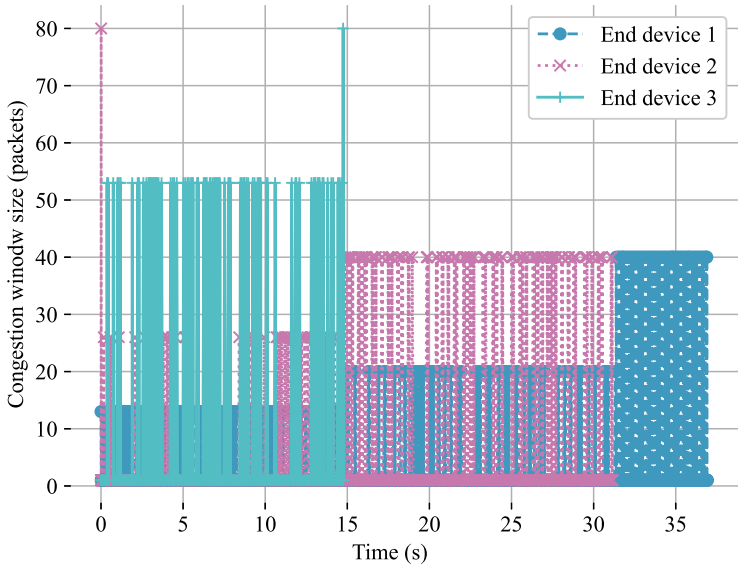


Figure 4.4: The behavior of cwnd of the RACC algorithm.

end devices 1, 2, and 3 at the same time using socket programming, to evaluate the airtime consumption and overall throughput achieved for CUBIC and RACC algorithms. The obtained results are discussed in the following section.

Since the performance of CUBIC CC algorithm is better than other existing conventional CC algorithms in terms of throughput and RTT fairness, the performance of designed RACC algorithm is validated against CUBIC algorithm only [24].

4.7.2 Results and discussion

On sending the fixed payload of 5 MB from the end devices using the designed RACC algorithm, the $cwnd$ of each end device was recorded and plotted in Figure 4.4. From the plot, it can be seen that the $cwnd$ is dependent on the physical data rate of the end device. On starting the TCP connection at the same time, the End device 3 with a higher physical data rate has a higher $cwnd$ value. Further at the 15th second on the plot, it can be noticed that the End device 3 finishes its data transfer and the $cwnd$ of End device 1 and End device 2 are updated. A similar effect can be seen at the end of the transfer from End device 3, where $cwnd$ of End device 2 gets increased to utilize the released capacity. Further, when the transfer from End device 2 is finished, the $cwnd$ of End device 1 increases to utilize the whole capacity left.

For the same test of sending a fixed 5 MB payload, the throughput of each

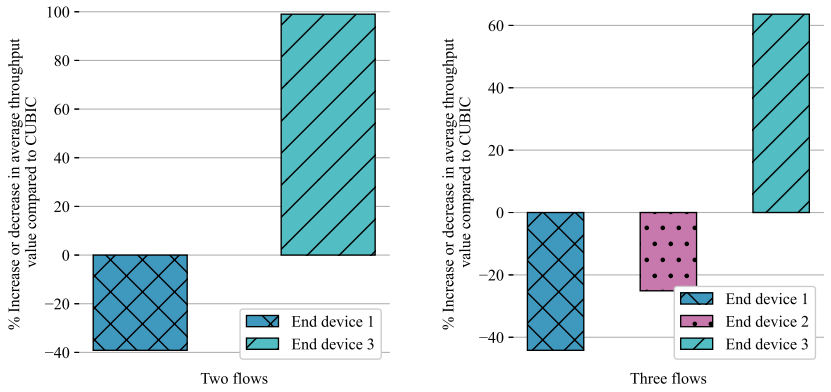


Figure 4.5: Percentage increase or decrease in average throughput of the RACC algorithm as compared to the CUBIC.

end device was recorded. The test was conducted for two flows (End device 1 and End device 3) and three flows (End device 1, End device 2, and End device 3). The obtained results in Figure 4.5 indicate the percentage increase or decrease in throughput of each end device as compared to the CUBIC CC algorithm. For the case of two flows, it can be noticed that there is a reduction in the throughput of End device 1 as compared to the throughput of the CUBIC algorithm. At the same time, there is around 99% improvement in the throughput of End device 3 as compared to the CUBIC algorithm. For the case of three flows, the throughput of End device 1 and End device 2 is lower than that of the CUBIC algorithm. End device 3 with a high physical data rate has increased its throughput by 45% as compared to the performance in the CUBIC algorithm.

The overall improvement in the throughput for End device 3 in case of three flows is lower than that of two flows, the reason being, that in the case of two flows, the number of devices contending for the channel is less in number. Hence, when one of the devices finishes its transfer, the other device can fully utilize the network. Whereas, in the case of three nodes, after one of the devices finishes its transfer, the other two nodes still contend for the channel access. This results in more white spaces as compared to the case of two nodes. This effect can be seen in both CUBIC and RACC algorithms, but the effect is higher in the case of RACC. Therefore, to answer the first question of the problem statement, the high physical data rate device has 99% and 45% better throughput on using the RACC algorithm than the CUBIC algorithm when there are two and three flows in the network respectively. On the other hand, the low physical data rate end devices can be penalized for their throughput when using RACC, as was the case for End device 1 and End device 2. However, this is a trade-off for better airtime fairness usage between end devices.

Further, the airtime consumed by each end device is calculated and plotted the instantaneous airtime by summation of the packet air times over a 500 ms timeslot. From Figure 4.6, it can be noticed that airtime consumption is inversely proportional to the physical data rate of the end device. Since the CUBIC algorithm does not consider airtime fairness in the design, all the end devices finish the data transmission of a fixed 5 MB payload at the same time. Whereas, since the RACC algorithm is designed with the intention to achieve airtime fairness, this was tested by continuously sending the data for a fixed time of 40 s and plotting the airtime calculated by AP in Figure 4.7. From the plot, it can be noticed that the airtime consumption of all three end devices lies in the range of 5-30 ms. The calculated airtime on sending the fixed payload of 5 MB is plotted in Figure 4.8 and the airtime of the three devices lie in the range of 10-40 ms, with the added advantage of improved performance in terms of throughput. When End device 3 finishes its data transfer, the airtime of End device 1 and End device 2 is slightly increased and lies in the range of 20-40 ms. Finally, when End device 2 finishes its transfer, the airtime consumption of End device 1 is increased further.

Using these values, the percentage difference between the airtime consumption was calculated for the devices with low and high physical data rates, respectively. Since End device 1 has the lowest physical data rate, the calculated percentage difference between the airtimes of End device 1 and End device 2 and the percentage difference between the airtimes of End device 1 and End device 3 for CUBIC and RACC algorithms and plotted in Figure 4.9. From the plot, it can be seen that in the case of CUBIC, the percentage difference between the airtime is as high as 119%. This percentage difference is between End device 1 and End device 3, i.e., between the lowest physical data rate and highest physical data rate devices of the network. Whereas, in the case of the RACC algorithm, the percentage difference is as low as 15% and 30% on average. Therefore, to answer the second question of the problem statement, the percentage difference in the airtime consumption of a high physical data rate device is 30% on average when the RACC algorithm is used.

Though the percentage difference in airtime of low and high physical data rate devices is as low as 15%, there is still room for improvement. From Figure 4.4 it can be noticed that there are still a few gaps between the $cwnd$ of End device 2 and End device 3. Also, the End device 1 does not fully utilize the available bandwidth when there are no other flows in the network. Since the feedback from AP is explicit, the designed RACC reacts immediately to this and does not take into account the events in the past nor predict the future. Hence there is a possibility of underutilization or overutilization of the wireless bandwidth by certain devices.

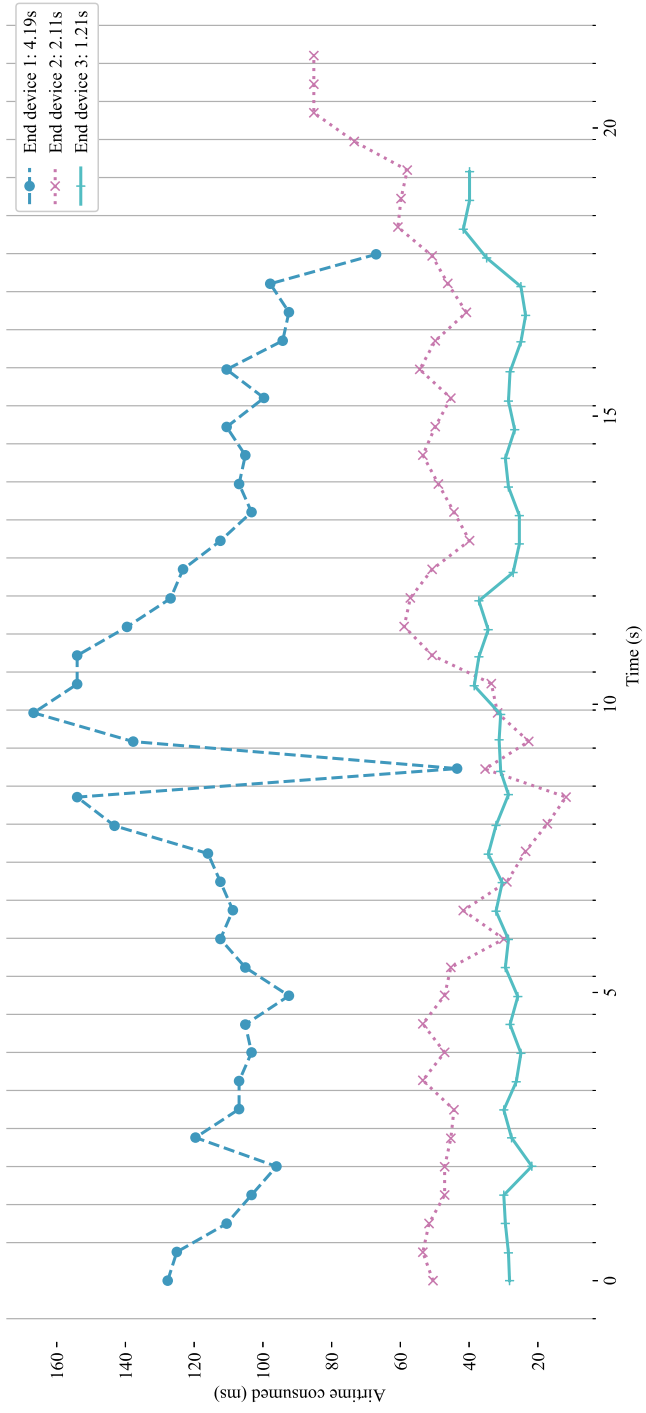


Figure 4.6: Instantaneous airtime of the CUBIC algorithm on sending a fixed data payload of 5 MB.

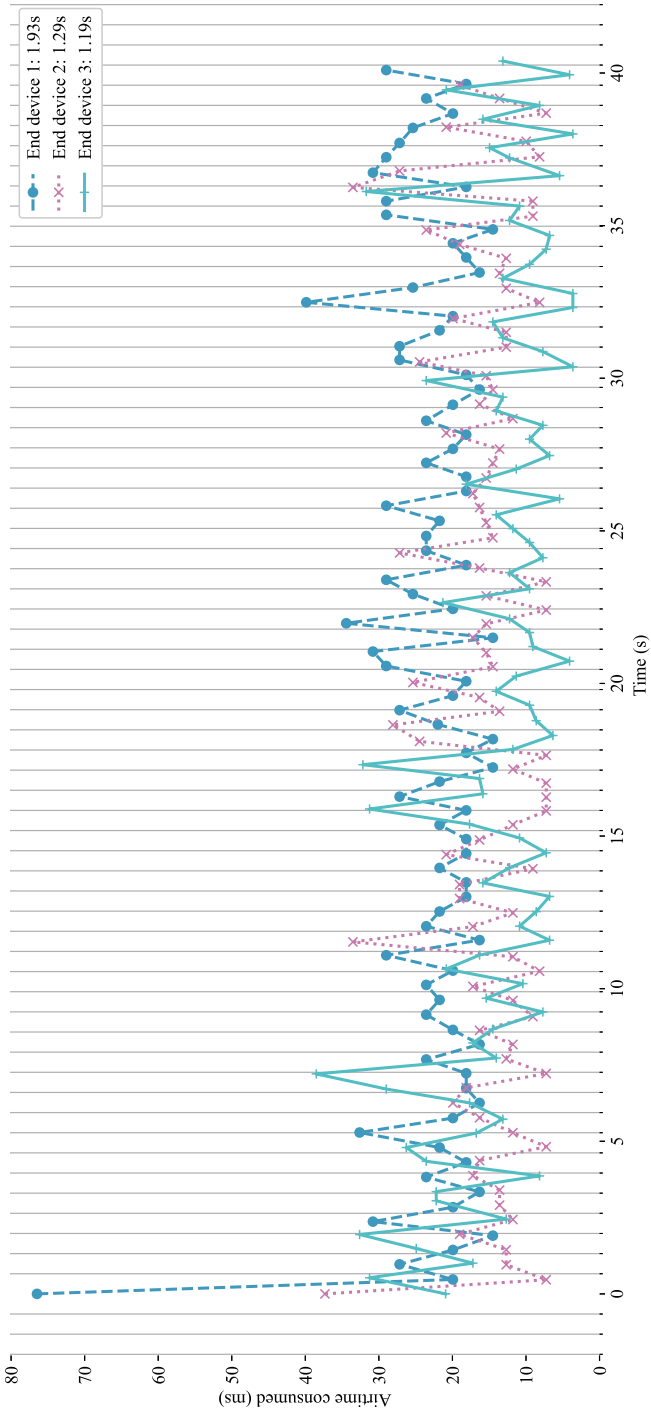


Figure 4.7: Instantaneous airtime of the RACC algorithm on sending data for 40 s.

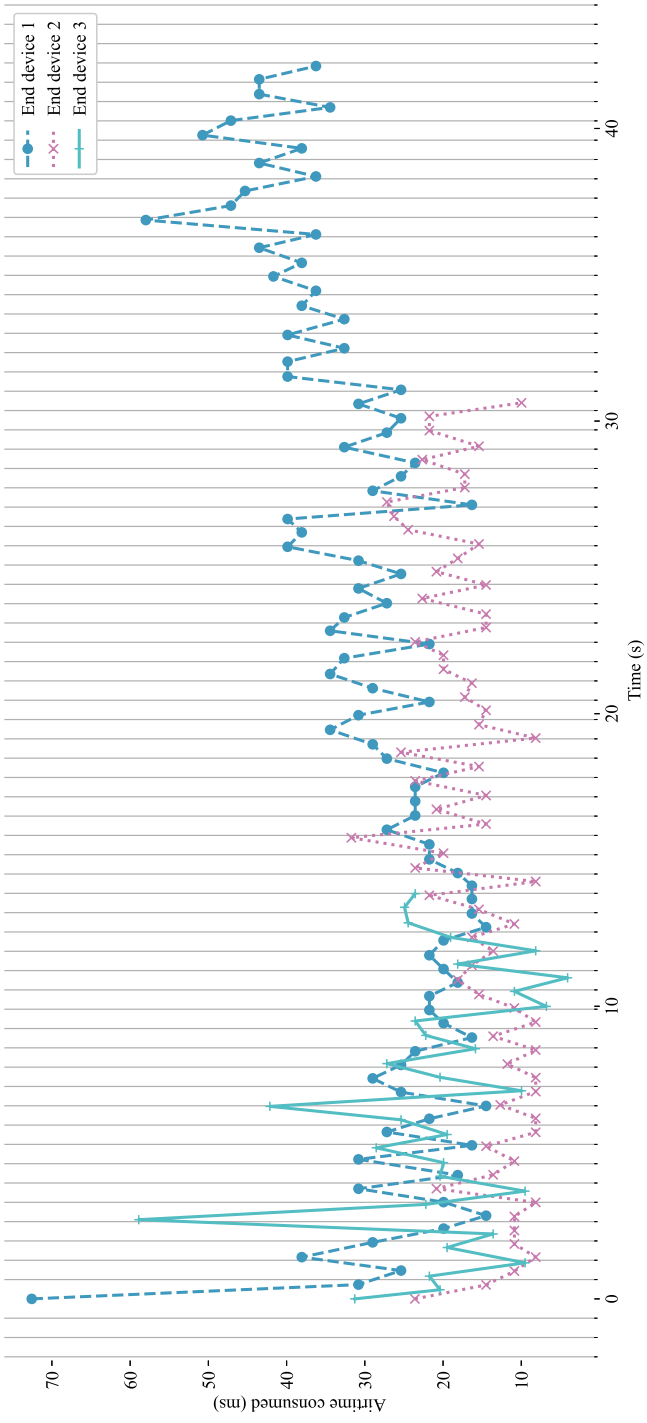


Figure 4.8: Instantaneous airtime of the RACC algorithm on sending a fixed data payload of 5 MB.

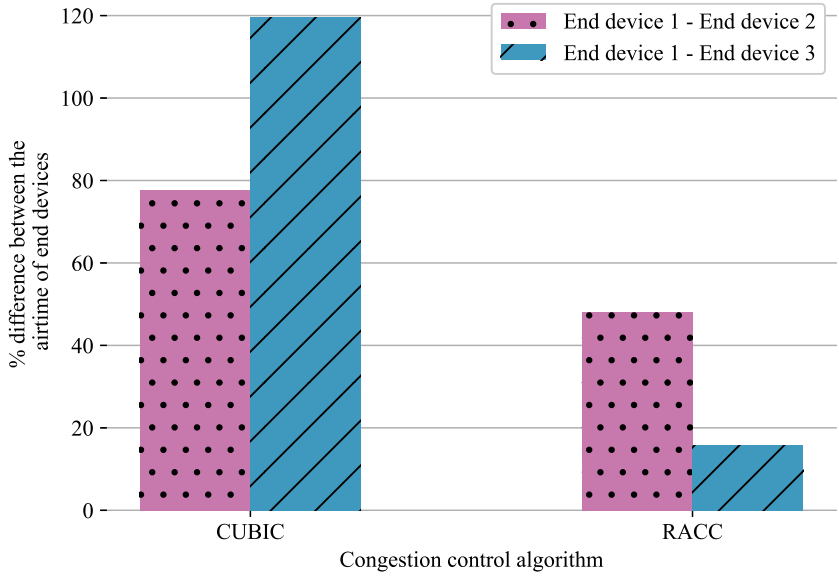


Figure 4.9: Percentage difference in airtimes of low and high physical data rate devices.

4.8 Conclusion

With a large number of industrial applications moving towards Wi-Fi, there is a requirement for flexible, real-time, reliable services from Wi-Fi protocols. Since existing application and transport layer protocols are designed for wired networks, it is important to address these drawbacks in wireless scenarios. One such important drawback is throughput degradation in mixed physical data rate networks caused by airtime unfairness between low and high physical data rate devices. The designed RACC algorithm adapts the application data transfer rate based on the real-time network and airtime feedback, thus solving the issue of throughput degradation in mixed physical data rate industrial wireless networks. The implemented design was benchmarked against the CUBIC CC algorithm. The high physical data rate device had an average of 72% improvement in its throughput as compared to CUBIC when the RACC algorithm was used. The percentage difference between the airtime of the lowest and the highest physical data rate devices was reduced to as low as 15%, with an average of 30%. One of the main advantages of this design is that it was implemented on COTS devices as compared to the other works that were carried out in simulators.

The algorithm NACC, designed in Chapter 3, focuses on listening to the application demands and catering to their needs, unlike providing the fixed quality of service as it is in the current state of the art. The algorithm additionally considers the

real-time network state in deciding the $cwnd$, making it robust to the unpredictable changes in the wireless network environments. The main application of this CC algorithm is to provide differentiated services to those flows that are not catered to by slicing and scheduling in wired-wireless time-sensitive networks. But the algorithm RACC is mainly intended to achieve airtime fairness in Wi-Fi networks. Instead of the aggregated flow information, aggregated airtime information is shared by AP to the end device. Unlike the aggregated flow information, aggregated airtime information is shared on a per-device basis.

References

[1] A. M. Abdul-Hadi, O. Tarasyuk, A. Gorbenko, V. Kharchenko, and T. Hollstein. *Throughput estimation with regard to airtime consumption unfairness in mixed data rate Wi-Fi networks*. Communications-Scientific letters of the University of Zilina, 16(1):84–89, 2014.

[2] F. Safdari and A. Gorbenko. *Experimental Evaluation of Performance Anomaly in Mixed Data Rate IEEE802. 11ac Wireless Networks*. In 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT), pages 82–87. IEEE, 2019.

[3] F. Safdari and A. Gorbenko. *Theoretical and experimental study of performance anomaly in multi-rate IEEE802. 11ac wireless networks*. Radioelectronic and Computer Systems, (4):85–97, 2022.

[4] F. Peng, B. Peng, and D. Qian. *Performance analysis of IEEE 802.11 e enhanced distributed channel access*. IET communications, 4(6):728–738, 2010.

[5] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke. *Tighter application-network interfacing to drive innovation in networked systems*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 53–57, 2021.

[6] 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 Transactions on Network and Service Management, 18(1):627–641, 2020.

[7] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. *Restructuring endpoint congestion control*. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pages 30–43, 2018.

[8] J. Hoblos. *New Adaptive 802.11 MAC Protocol to Enhance Throughput and Fairness in Multihop Wireless Networks*. In 2021 International Wireless Communications and Mobile Computing (IWCMC), pages 1908–1913. IEEE, 2021.

[9] J. Kim, H. Jin, J.-B. Seo, S. H. Kim, and D. K. Sung. *Adaptive transmission control for uplink/downlink fairness in unsaturated CSMA networks*. IEEE Communications Letters, 23(10):1879–1882, 2019.

[10] A. Baiocchi. *Maximizing the stable throughput of heterogeneous nodes under airtime fairness in a CSMA environment*. Computer Communications, 210:229–242, 2023.

[11] J. Lei, J. Tao, J. Huang, and Y. Xia. *A differentiated reservation MAC protocol for achieving fairness and efficiency in multi-rate IEEE 802.11 WLANs*. IEEE Access, 7:12133–12145, 2019.

[12] S. I. Yu and C. Y. Park. *A Responsible Airtime Approach for True Time-Based Fairness in Multi-Rate WiFi Networks*. Sensors, 18(11):3658, 2018.

[13] Z. Mao and Y. Jiang. *Fair airtime allocation for content dissemination in WiFi-direct-based mobile social networks*. In 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), pages 1–7. IEEE, 2017.

[14] Y. Fang, B. Doray, and O. Issa. *A practical air time control strategy for Wi-Fi in diverse environment*. In 2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), pages 1–6. IEEE, 2017.

[15] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, and A. Brunstrom. *Ending the Anomaly: Achieving Low Latency and Airtime Fairness in {WiFi}*. In 2017 USENIX Annual Technical Conference (USENIX ATC 17), pages 139–151, 2017.

[16] D. J. Kulenkamp and V. R. Syrotiuk. *Support for differentiated airtime in wireless networks*. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 1–6. IEEE, 2021.

[17] L. Tokar and Y. Krasnozheniuk. *Analysis of Airtime Fairness Technology Application for Fair Allocation of Time Resources for IEEE 802.11 Networks*. In International Scientific-Practical Conference" Information Technology for Education, Science and Technics", pages 635–655. Springer, 2022.

[18] C. Ben Ameur, E. Mory, and B. Cousin. *Combining traffic-shaping methods with congestion control variants for HTTP adaptive streaming*. Multimedia Systems, 24:1–18, 2018.

[19] J. Jayaudhaya, S. Supriya, V. A. Kandaswamy, S. Pandi, S. Kamatchi, and C. P. Priya. *Acoco: an adaptive congestion control approach for enhancing coap performance in iot network*. In 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), pages 189–194. IEEE, 2023.

[20] T. K. Mishra, K. S. Sahoo, M. Bilal, S. C. Shah, and M. K. Mishra. *Adaptive congestion control mechanism to enhance TCP performance in cooperative IoV*. IEEE Access, 11:9000–9013, 2023.

[21] G. Bianchi, L. Fratta, and M. Oliveri. *Performance evaluation and enhancement of the CSMA/CA MAC protocol for 802.11 wireless LANs*. In Proceedings of PIMRC'96-7th International Symposium on Personal, Indoor, and Mobile Communications, volume 2, pages 392–396. IEEE, 1996.

[22] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *CUBIC for fast long-distance networks*. Technical report, 2018.

[23] W. Eddy. *Rfc 9293: Transmission control protocol (tcp)*, 2022.

[24] S. Patel, Y. Shukla, N. Kumar, T. Sharma, and K. Singh. *A comparative performance analysis of tcp congestion control algorithms: Newreno, westwood, veno, bic, and cubic*. In 2020 6th International Conference on Signal Processing and Communication (ICSC), pages 23–28. IEEE, 2020.

5

Feedback-based Control loop Congestion Control Algorithm for Wireless Networks

In the previous chapter, a reactive CC algorithm was designed to address the issue of sirtime unfairness. The RACC is designed to respond re-actively to specific feedback from AP, disregarding past events and future predictions. As a result, it's possible that some devices will use the wireless bandwidth excessively or insufficiently. In this chapter, a feedback-based PI control loop CC technique is designed to tackle the problem of throughput deterioration in Wi-Fi networks caused by fair channel access. To consider the past and future events, the intermediate node is set up to monitor each end device's airtime consumption and transmit the total airtime data. The designed CC algorithm utilizes this information to calculate the error in airtime usage, thus correcting the number of data packets to be shared during next data transfer. By adjusting the data transfer rate of the applications, the designed algorithm, in contrast to the existing CC algorithms, provides airtime fairness and is resilient to changes in network parameters and channel quality.

⋆ ⋆ ⋆

This chapter is based on:
Feedback-based Control loop Congestion Control Algorithm for Wireless Networks

Published in IEEE Access, July 2024.

⋆ ⋆ ⋆

5.1 Introduction

Wi-Fi plays a crucial role in connecting private-professional networks by providing varying data rates based on channel quality. Despite advancements in Wi-Fi protocols, there is still a problem of throughput degradation for higher data rate devices when lower data rate devices are around, due to airtime unfairness. The diverse requirements of emerging applications emphasize the necessity for network protocols that cater specifically to these needs. To tackle the problem of throughput degradation in devices that use higher physical data rates, a reactive and adaptive feedback-based CC algorithm was designed in the previous chapter. The algorithm adapted its application data transfer rate based on the real-time network context and airtime allocation feedback from the AP.

Despite achieving improvement in throughput of higher physical data rate end devices and fairness in airtime among all the end devices, the benchmark results against the CUBIC CC algorithm indicated that the reactive algorithm used the wireless channel inefficiently. Since the feedback from the AP was explicit in nature, the designed algorithm reacts immediately to this and does not take into account the events in the past nor predict the future. As a result, it's possible that some devices will use the wireless bandwidth excessively or insufficiently. Therefore, this chapter discusses the design of a proactive CC algorithm that decides the application data transfer rate based on the feedback from AP obtained by continuous monitoring of airtime consumed by each device.

The problem of throughput degradation in mixed physical data rate Wi-Fi networks is thus addressed by designing a Feedback-based Control loop CC algorithm (FCCC). The aim of this work is to monitor the airtime used by each end device by correcting the $cwnd$ based on the aggregated airtime feedback from AP. The key contributions of this chapter are,

1. Design of an aggregated airtime feedback system based on airtime usage in AP.
2. Design of proactive algorithm, FCCC.
3. Performance evaluation of the designed CC algorithm in comparison with CUBIC in terms of airtime fairness, throughput.
4. Robustness of the designed algorithm to changing network conditions.

The goals of the chapter and the essential frameworks are explained in Section 5.2. The design and implementation of aggregated airtime feedback system

in AP are elaborated in Section 5.3. Section 5.4 provides a detailed explanation of the design and implementation of FCCC for wireless networks. The performance of FCCC algorithm is explained and the research outcomes are discussed in Section 5.5. Finally, Section 5.6 concludes this chapter.

5.2 Goals and essential frameworks

This chapter elaborates on the design of FCCC algorithm that improves the throughput of end devices with higher physical data rates in wireless networks based on the aggregated airtime feedback from AP. To achieve this, the CC algorithm should consider **past events** and **foresee** the airtime usage, hence should be **proactive**. The CC algorithm should be **stable**, i.e., have a stable application data transfer rate to achieve better throughput. The algorithm should be **robust** to changing physical data rates and other network conditions. The actions taken by the end device are dependent on INT feedback on real-time network conditions as well as aggregated airtime feedback from AP. Since the end devices are unaware of the other devices in the network, it is the responsibility of AP to give the **aggregated airtime feedback** considering the ongoing changes in the network also ensuring the **privacy** of other end devices.

To design FCCC, the transport layer of the end device should be updated with the INT and aggregated airtime feedback from AP. The end device should be able to recalculate and assign the $cwnd$ based on this feedback. These operations are facilitated by the following frameworks:

1. **INT for wireless networks:** The INT framework for wireless networks designed in [1] is a low overhead, continuous network monitoring technique that collects real-time network states such as physical data rate, RSSI, buffer capacity, etc. from the intermediate network nodes. The intermediate network nodes add the network state information as an IPv6 extension header which can be exploited to add the aggregated airtime information in the designed feedback-based system.

2. **APP-NET interaction API:** The real-time network state information obtained from INT is extracted in the network layer of the end device. The APP-NET API designed in [2] provides an interface where an application can publish its service requirements and the network layer can publish the real-time network information collected using INT. This interface is extended to the transport layer in [3] resulting in application-transport-network layer interaction. In the designed system, this API is used to publish the INT and aggregated airtime feedback from AP to the transport layer of the end device.

3. **The CC plane (CCP):** The CCP framework designed in [4] allows the reconfiguration of CC algorithms. The framework allows the implementation of

algorithms in Python programming language which adds additional flexibility to the design.

5.3 Aggregated airtime feedback system in AP

Using INT, APs can now update the end devices connected to them with the real-time network information with additional aggregated information about other network flows. In this section, the design and implementation of the aggregated airtime feedback system in AP is elaborated. Figure 5.1 gives an overview of the network architecture with the additional processing in AP. The meaning and unit of each terminology used in the coming sections is listed in Table 5.1.

5.3.1 Relevant monitoring parameters

Physical data rate (dr) directly affects the airtime consumed during a data transfer. Therefore, knowing the dr of the link is essential for the FCCC algorithm to predict future airtime usage. Since the goal is to adapt to the changing network conditions, the FCCC must also be aware of the number of end devices (dc) in the network. Hence from the available wireless telemetry options, dr, dc, and $loss_w$ are the monitoring parameters that are considered for the design. FCCC gets real-time information on these parameters from AP.

5.3.2 Bookkeeping in AP

Since the wireless channel is continuously changing, AP calculates the statistics of airtime usage over a fixed period called timeslot (ts) to get a perspective on the total airtime usage of each end device using the channel. The methodology used to dynamically set the ts is explained in Section 5.3.4. To provide aggregated airtime feedback, AP maintains an airtime consumption table for the end devices connected to it. The end device is added to the table every time it initiates a TCP connection i.e., through the SYN packet. Every time the AP receives a packet from the sender end device, the AP calculates the airtime consumed for the data transfer ($airtime$) using Equation 5.1:

$$airtime = T_{PHY} + \frac{bits_s + bits_t + N * 8 * (h_{MAC} + h_{LLC} + p_{IP})}{dr} \quad (5.1)$$

where T_{PHY} indicates the time of preamble and signal, $bits_s$ and $bits_t$ are service and tail bits respectively, and N indicates the number of aggregated packets (in this case, 1), h_{MAC} and h_{LLC} indicate the MAC and LLC header sizes respectively in bytes, and p_{IP} indicates the IP packet size in bytes. The MAC layer overheads of Distributed Interframe Spacing (DIFS) (T_{DIFS}), Short Interframe Spacing (SIFS)

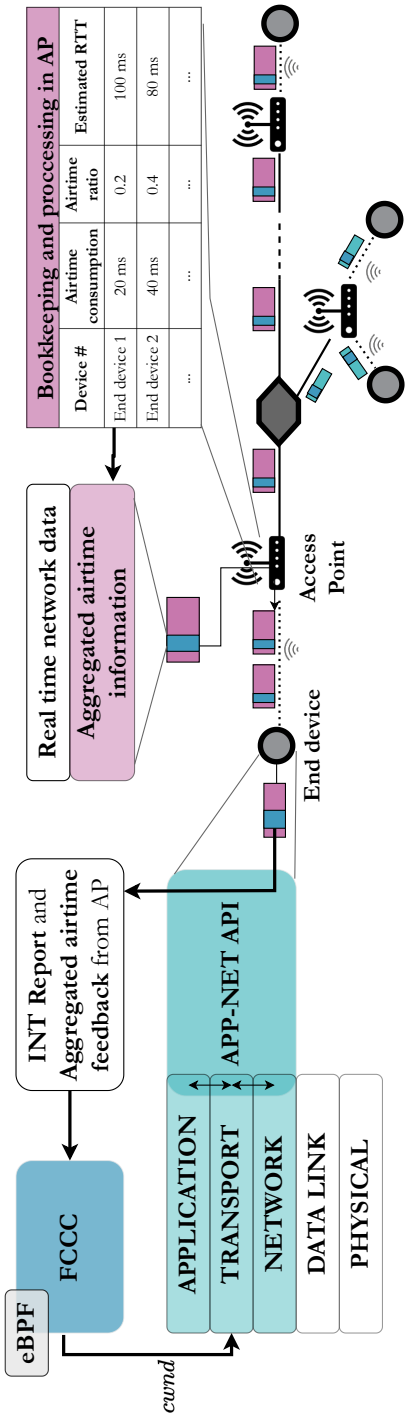


Figure 5.1: Network architecture overview with additional processing at AP.

Table 5.1: Summary of abbreviations, their meanings and units

Parameters at AP

Parameter	Meaning	Unit
dr	Physical data rate of an end device recorded by AP	Mbps
dc	Device count, Number of end devices using the channel	none
ts	Timeslot, time period over which the AP calculates the aggregated airtime usage	second
t_p	Airtime of packet	second
p_{ip}	IP payload	byte
sp	Percentage of ts that can be used by the device including the margin for new connection initiation	%
ar	Aggregated airtime feedback calculated by AP over ts	%
w_ar	Weighted average of aggregated airtime feedback calculated by AP over ts	%
$loss_w$	Wireless packet loss	packet

Fixing timeslot at AP

Parameter	**Meaning**	**Unit**
$ertt$	RTT estimated by AP	second
$e2a_l$	Latency between end device to AP	second

Parameters at end device

Parameter	**Meaning**	**Unit**
t_p	Airtime of packet	second
c_p	Correction factor	none
$cwnd_{up}$	Congestion window upper bound	packet
$error$	Error between the aggregated airtime feedback and target sp	none
K_p	Proportional constant of PI controller	none
K_i	Integral constant of PI controller	none
e_p	Proportional error of PI controller	none
e_i	Integral error of PI controller	none
PI_{error}	The summation of proportional and integral error in PI controller equation	none
$cwnd$	Congestion window size	packet
$cwnd_{prev}$	Congestion window size of previous data transfer	packet

Parameters from kernel using eBPF

Parameter	**Meaning**	**Unit**
$loss_p$	Packets lost due to congestion in the network	packet
$bytes_{ACK}$	Bytes acknowledged by the receiver	byte
$bytes_{SACK}$	Bytes selectively acknowledged by the receiver	byte
p_{size}	Size of the packet	byte
$flow_{timeout}$	Flow timeout notification	boolean
MSS	Maximum Segment Size	byte

(T_{SIFS}), and time for MAC layer ACK packet (T_{ACK}) are also added to $airtime$ to get the t_p, as shown in Equation 5.2.

$$t_p = airtime + T_{DIFS} + T_{SIFS} + T_{ACK} \tag{5.2}$$

The t_p recorded in a timeslot (ts_{t-1}, ts_t) is summed over to calculate the aggregated airtime ratio (ar) using Equation 5.3

$$ar_t = \frac{\sum_{i=1}^{n} t_{p_i}}{ts_t}, t = |ts|, n = |t_p| \text{ recorded in } (ts_{t-1}, ts_t) \tag{5.3}$$

where t indicates the number of ts blocks over time.

The calculated ar is updated in the airtime consumption table for every ts period. To give a network overview to the end device, the AP assigns a setpoint (sp) value to each end device using Equation 5.4, which is the percentage of ts that can be utilized by an end device with a 20% margin for initiation of a new connections. A margin of 20% is sufficient for an end device to start a new connection[1].

$$sp = \frac{0.8}{dc} \tag{5.4}$$

Since the aim is to achieve equal airtime fairness among all the end devices, the sp is assigned equally.

5.3.3 Stabilizing the aggregated airtime feedback

The ar feedback calculated by the AP at every ts is instantaneous feedback of the airtime usage. Making decisions based on instantaneous value can result in highly fluctuating outcomes. To avoid this the weighted average of ar of previous timeslots is averaged with the current aggregated airtime ratio for each end device. For ar values over t timeslots, the weighted average of ar, i.e., w_ar (in future referred as aggregated airtime feedback) is calculated using Equation 5.5.

$$w\_ar_t = 2^{-t} * ar_1 + \sum_{i=2}^{t} 2^{(i-1)-t} * ar_i \tag{5.5}$$

This average value is calculated for each end device and stored in the airtime consumption table. The w_ar is sent as feedback after each timeslot.

[1]Wi-Fi traffic distribution between control, management and data frames: https://www.itweb.co.za/article/unwrapping-wifi-6-getting-out-of-the-traffic-jam/o1Jr5qx96jDvKdWL

5.3.4 Dynamic assignment of ts in AP

Wireless channels are continuously changing at each instant of time, therefore, it is essential to discretize and look at the airtime usage of each end device over a fixed ts to get a better outlook on the network state. Especially in wireless networks, the RTT is not fixed, and it depends mainly on the channel quality and the processing time in intermediate nodes of the network, therefore, the ts value is assigned dynamically.

The aggregated airtime information sent by AP is received by the sender end device through the ACK or Selective ACK (SACK) packet. The value of ts should be such that it includes at least one RTT of all the end devices using the channel at every instant of time. A very small ts value can result in missing feedback of certain end devices, whereas very large ts can result in delayed feedback and be ineffective to the changing network conditions. AP thus keeps track of the maximum RTT of each end device and dynamically assigns the ts to the maximum value of this range using Equation 5.6. In Equation 5.6 for n end devices using the wireless channel at each instant of time, $ertt$ is the RTT estimated by AP between the AP and receiver and $e2a_l$ is the one-way latency between the sender end device and AP. $e2a_l$ is estimated using the *rx_timestamp* of INT telemetry option. The *rx_timestamp* is added by the sender to the INT header. This is the timestamp at which the packet is generated at the sender. This can be subtracted from the current timestamp at the AP to obtain $e2a_l$.

$$ts = max((ertt + 2 * e2a_l)_1, (ertt + 2 * e2a_l)_2, ...(ertt + 2 * e2a_l)_n) \quad (5.6)$$

Thus, AP sends ts and sp (instead of dc) values as feedback in the IPv6 extension header along with other wireless telemetry fields such as dr and $loss_p$ in every TCP packet received from the sender end device whereas, the aggregated airtime ratio, w_ar, information is sent only after every ts seconds. The INT information is added by the AP extracted by the receiver end device and sent as feedback through the ACK/SACK packet to the sender end device. In case of multiple INT information packets, the receiver adds the latest INT information as feedback. The aggregated airtime ratio along with ts and sp ensures that end devices get a perspective of their airtime usage and ongoing activities in the network without compromising the privacy of other end devices. After the completion of a TCP connection, the table entry of the end device is removed from the airtime consumption table in AP.

The modifications in the AP are implemented using the Click modular router framework[2].

The airtime usage is decided on a per-end device basis and not a per flow basis. Therefore, from the AP's point of view, irrespective of number of flows from the end

[2]Click modular router framework documentation: https://github.com/kohler/click

device, the aggregated airtime ratio is calculated for each end device. Though the current implementation focuses on achieving equal airtime fairness, the parameters monitored in AP (such as sp) and APP-NET allow priority-based airtime allocation in the future.

5.4 Design and implementation of FCCC

In this section, design and implementation of the FCCC algorithm is elaborated. The designed algorithm uses INT and aggregated airtime feedback from AP to decide the $cwnd$ at each instant of time.

5.4.1 Control loop-based FCCC design

The goal of achieving airtime fairness, thus improving the throughput of end devices with higher physical data rate is considered by AP by sending the aggregated airtime feedback (w_ar), sp, ts, and dr as feedback to the end device. It is the job of the CC algorithm in the end device to analyze the previous airtime usage, foresee the future airtime usage, and make the right correction in the $cwnd$ whilst maintaining a stable data transfer for higher throughput. To achieve these goals, the algorithm needs to know its performance i.e., the over- or under-usage of the airtime in a timeslot in terms of number of packets. Subtracting the amount of foreseen airtime usage by the previous airtime usage over a given timeslot (dividing the difference by the timeslot value) gives the correction factor (c_p), i.e., the number of packets to be added to or subtracted from previous data transfer to achieve intended airtime usage as shown in Equation 5.7.

$$c_p = cwnd * \frac{t_p}{ts} - cwnd_{prev} * \frac{t_p}{ts} \tag{5.7}$$

where t_p is the time per packet calculated in the end device using Equation 5.2. (The assumption is $ts_t \approx ts_{t-1}$, since the difference between the two values over a period of time in negligible.)

To achieve the above mentioned goals, the FCCC has been designed as a loop-based Proportional Integral (PI) controller. Therefore, the PI_{error} is the outcome of the PI controller (Equation 5.8):

$$PI_{error} = K_p * error + K_i * \int_0^t error(t)dt \tag{5.8}$$

where $error = sp - w\_ar$ and $K_p = sp, K_i = 0.1 * sp$. The $error$ in Equation 5.8 indicates the deviation in the airtime consumed in the previous timeslot as compared to the allocated airtime, sp, sent as feedback by AP. w_ar is the outcome of previous

congestion window size ($cwnd_{prev}$). Hence c_p can be derived also as the fraction of $cwnd_{prev}$ by calculating the PI_{error} from the sp:

$$c_p = \frac{PI_{error}}{w\_ar} * cwnd_{prev} \tag{5.9}$$

The proportional error with proportional gain K_p determines the rate of direct response to the error, whereas integral error is the summation of the errors since the beginning of the connection, with integral gain K_i, eliminating the steady-state error linked with proportional response. The gains achieved with K_p and K_i should be in sync with the sp set by AP and thus proportional to the sp. The values of K_p and K_i were tuned to sp and $0.1 * sp$ respectively. Substituting the values, the $cwnd$ of the FCCC algorithm can be obtained using Equation 5.10. Since the $cwnd$ is the unknown factor, combining and restructuring Equation 5.7 and Equation 5.9, the $cwnd$ of the current data transfer can be calculated.

$$cwnd = (PI_{error} * \frac{ts}{t_p}) * \frac{cwnd_{prev}}{w\_ar} + cwnd_{prev} \tag{5.10}$$

$$[PI_{error} * \frac{ts}{t_p}] \in [-sp, sp]$$

By calculating the $error$ for each timeslot, the CC algorithm foresees future airtime usage. Since the $error$ is calculated from sp, the algorithm considers the other flows in the network. These features make the algorithm proactive in nature. On starting the connection, the proportional gain in the equation gives the instant gain of the $cwnd$ value thus achieving the highest possible throughput. The integral error of the equation makes the algorithm respond quickly by considering the previous airtime usage and the duration of the error. The fast response of the integral gain can also result in instability and oscillation of the overall outcome, thus, the e_i of the $cwnd$ equation is clamped between -1 and 1. Since the value of $error$ lies in the range of $-sp$ and sp, $PI_{error} * ts/t_p$ is also clamped, which in turn helps to maintain stable data transfer. The value of the $cwnd$ varies with the changing physical data rate through t_p, thus the algorithm is robust to the changing physical data rate. Though the outcome of the algorithm is stable and robust, and foresees the future, the design does not ensure whether the airtime used by the calculated $cwnd$ is under the limitation imposed by the AP. Therefore, to achieve the last step of proactiveness, boundary condition to the $cwnd$ is applied. Figure 5.2 gives an overview of the FCCC algorithm.

Bounds to the $cwnd$: The airtime used by an end device should be less than or equal to the sp feedback from the AP, where airtime used over one timeslot is

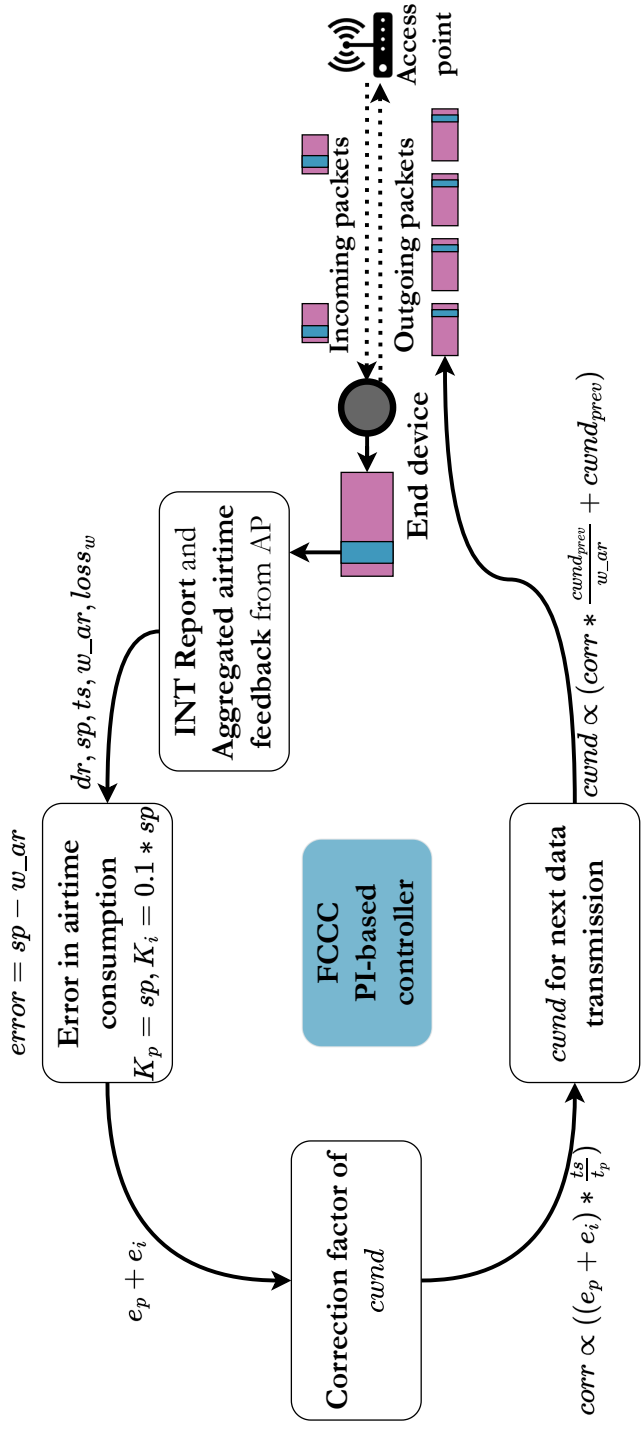


Figure 5.2: Overview of control-loop based CC algorithm based on network feedback.

obtained by the product of $cwnd$ and time per packet divided by the timeslot.

$$w\_ar \leq sp$$
$$cwnd \leq \frac{ts}{t_p} * sp$$

Writing the above relation in an equation gives the upper bound of the $cwnd$.

$$cwnd_{up} = \lfloor 1 + \frac{ts}{t_p} * sp \rfloor \tag{5.11}$$

The algorithm ensures that the value of the $cwnd$ is lower than the upper bound in Equation 5.11. The added upper bound foresees the airtime usage, ensures stability and the dependency on sp allows it to adapt to the changing network conditions such as increase in the number of end devices in the network.

The designed FCCC algorithm is represented in the Algorithm 3. On starting a new TCP connection, the FCCC sends a SYN packet with $cwnd$ value equal to $cwnd_{init}$. The value of $cwnd_{init}$ is equal to the $ssthresh$ value set in conventional CC algorithms. The algorithm receives TCP end-to-end information such as total bytes acknowledged ($bytes_{ACK}$), total bytes selectively acknowledged ($bytes_{SACK}$), number of packet losses (p_{loss}), size of the payload sent in a packet (p_{size}) and whether the connection is lost, and TCP enters connection timeout ($flow\_timeout$). In case of a timeout, the algorithm resets the variables as mentioned in $reset\_function()$. If the connection is successful, the device also receives feedback from AP such as sp, ts, w_ar and real-time network information such as dr and packet loss due to wireless conditions ($loss_w$) through INT.

5.4.2 Implementation of the FCCC

On receiving every ACK packet, the algorithm calculates the time per packet and congestion window upper bound based on the feedback from AP and enters the $on\_ack()$ function. The algorithm calculates the $cwnd$ value using Equation 5.10 for every w_ar feedback received from the AP. The algorithm also checks for the upper boundary condition based on the current feedback from the network. In case of no feedback, the $cwnd$ value is kept the same as the previous congestion window size.

In the case of a p_{loss} notification from the kernel, the algorithm checks whether the packet loss is due to wireless conditions such as interference, poor channel quality, obstacles in the channel, etc., (represented as $loss_w$). This is obtained by finding the difference between the counter values in the INT header [1]. The $loss_w$ variable is published by APP-NET. In case of packet loss due to a wireless channel, the $cwnd$ is kept constant. In case of a packet loss due to congestion in the network, the conventional CUBIC CC algorithm reduces the value to 70% of the previous

Algorithm 3 The designed FCCC algorithm

Initialization:
$reset\_function()$
$cwnd \leftarrow cwnd_{init}$,
Value reported by eBPF: MSS
$cwnd_{out} \leftarrow MSS * cwnd$

$on\_report()$:
Values updated from INT feedback:
dr, sp, w_ar, ts, $loss_w$
Values reported by eBPF:
$bytes_{ACK}$, $bytes_{SACK}$, $loss_p$, p_{size}, $flow\_timeout$, MSS
if $bytes_{ACK}$ or $bytes_{SACK}$ **then**

$$t_p \leftarrow \frac{bits_s + bits_t + 8 * (h_{MAC} + h_{LLC} + p_{size})}{dr} + T_{PHY} + T_{DIFS} + T_{SIFS} + T_{ACK}$$

$$cwnd_{up} \leftarrow \lfloor 1 + \frac{ts}{t_p} * sp \rfloor$$

    $on\_ack(sp, w\_ar, ts, t_p, cwnd_{up}, time_{prev}, cwnd_{prev}, MSS)$
else if $loss_p$ **then**
    $on\_loss(loss_w, MSS)$
else if $flow\_timeout$: **then**
    $on\_timeout()$
end if

$on\_clamp(value, lower, upper)$:
if $value \leq lower$ **then**
    $value \leftarrow lower$
else if $value \geq upper$ **then**
    $value \leftarrow upper$
end if
return $value$

28: $on\_ack(sp, w\_ar, ts, t_p, cwnd_{up}, time_{prev}, cwnd_{prev}, MSS)$:
29: **if** $w\_ar \neq 0$ **then**

$$error \leftarrow sp - w\_ar$$
$$dt \leftarrow current.time() - time_{prev}$$
$$e_p \leftarrow sp * error$$
$$e_i \leftarrow on\_clamp(e_i + (0.1 * sp * error * dt), -1, 1)$$
$$corr \leftarrow on\_clamp(((e_p + e_i) * \frac{ts}{t_p}), -sp, sp)$$
$$cwnd \leftarrow corr * \frac{cwnd_{prev}}{w\_ar} + cwnd_{prev}$$

30: **if** $cwnd \geq cwnd_{up}$ **then**
31: $cwnd \leftarrow cwnd_{up}$
32: **end if**
33: $cwnd_{prev} \leftarrow cwnd, time_prev \leftarrow current.time()$
34: **else**
35: $cwnd \leftarrow cwnd_{prev}$
36: **end if**
37: $cwnd_{out} \leftarrow MSS * cwnd$
38:
39: $on\_loss()$:
40: **if** $loss_w$ **then**
41: $cwnd \leftarrow cwnd_{prev}$
42: **else**
43: $cwnd \leftarrow cwnd_{prev} * \beta$
44: **end if**
45: $cwnd_{out} \leftarrow MSS * max(cwnd, cwnd_{init})$
46:
47: $flow\_timeout()$:
48: $reset\_function()$
49:
50: $reset\_function()$:
51: $cwnd_{init} \leftarrow 10$, $\beta \leftarrow 0.9$, $time_{prev} \leftarrow current.time()$, $cwnd_{prev} \leftarrow cwnd_{init}$,
52: $bytes_{ACK} \leftarrow 0$, $bytes_{SACK} \leftarrow 0$, $p_{loss} \leftarrow 0$, $p_{size} \leftarrow 0$, $dr \leftarrow 0$, $sp \leftarrow 0$,
53: $ts \leftarrow 0$, $loss_w \leftarrow 0$, $error \leftarrow 0$, $e_p \leftarrow 0$, $e_i \leftarrow 0$, $corr \leftarrow 0$,

value [5]. But in the case of FCCC, the performance of the algorithm was tested by reducing the value by 70%, 80%, 90%, and 95%. The FCCC performed better when the $cwnd$ was reduced to 90% of its previous value. Therefore, the value of the decrease factor β is set as 0.9.

The algorithm is implemented using Python programming language 3.8 with the CCP framework [4]. The end-to-end TCP information from the Linux kernel is made available to the algorithm by the CCP framework.

5.5 Results and Discussion

The performance of the designed FCCC algorithm was validated by defining Key Performance Indicators (KPIs) and conducting several tests to measure the improvement in throughput and airtime fairness achieved. The values of $cwnd$ and instantaneous airtime consumed are used as KPIs to measure the sensitivity to timeslot length. The improvement in the throughput and airtime fairness between the wireless end devices are the KPIs to validate the performance of the FCCC algorithm. The behavior of $cwnd$ and $cwnd_{up}$ for varying physical data rates are the KPIs to evaluate the robustness of FCCC algorithm to changing data rates. The instantaneous airtime consumption and $cwnd$ are the KPIs for the use case test with different sp values from AP. This section will elaborate on the test setup and the tests carried out to validate the performance.

5.5.1 Test setup

One of the drawbacks that is evident in the state of the art of CC algorithms innovation on top of Wi-Fi is the lack of implementation and experimentation in the commercially available wireless devices. Most of the research works are built and validated in a simulated environment. Therefore, the designed FCCC algorithm is implemented on a commercially available wireless node and the experiments are conducted in a w-iLab.2 testbed[3] on ZOTAC and DSS wireless nodes[4]. The experimental configuration includes a wired connection linking two APs (DSS nodes) who serve six end devices (ZOTAC nodes) in total, with three end devices linked to each AP. All the devices use Ubuntu 18.04 as operating system, with kernel version 4.15.0-45-generic, TCP version as per the standard in [6] and IP version 6. The APs are configured for 802.11n Wi-Fi standard and the aggregation is disabled. The airtime distribution is done on per-device basis. Allocation of airtime allotment among different flows in an end device is the task of the end device itself and is out of the scope of this work. Therefore, each end device has been restricted to one flow. To clearly see the performance of the designed algorithm, the physical

[3]imec iLab.t testbeds' documentation: https://doc.ilabt.imec.be/ilabt/wilab/

[4]w-iLab.2 hardware details: https://doc.ilabt.imec.be/ilabt/wilab/hardware.html#w-ilab-2-hardware

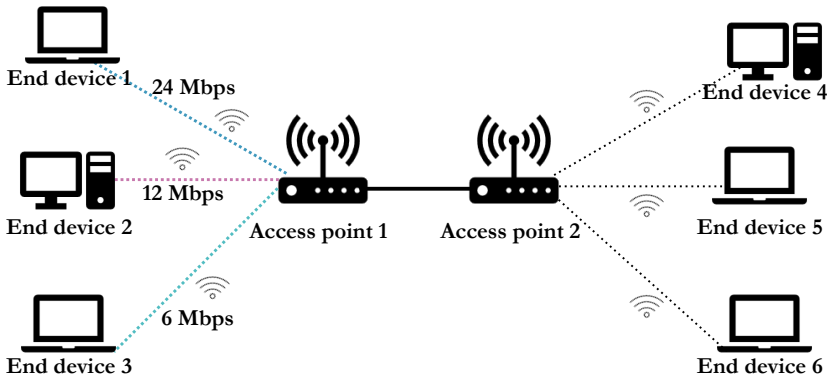


Figure 5.3: Wireless network setup used to test the sensitivity to ts length (Section 5.5.2) and compare the performance of FCCC and CUBIC (Section 5.5.3).

data rates of the end devices have been fixed to 6 Mbps, 12 Mbps, and 24 Mbps using `iwconfig` command for different tests. The INT is enabled in every packet to get the clear picture of ongoing changes in the network and evaluate the impact of FCCC algorithm accordingly. Computational overhead added by INT and the age of information is previously measured in [7] and it is shown to be limited.

5.5.2 Sensitivity of FCCC to timeslot length

The ts length after which the AP sends the w_ar feedback has a major impact on the performance of the FCCC algorithm, hence the sensitivity of the FCCC algorithm to different ts lengths was measured. The AP dynamically calculates the ts period based on the maximum end-to-end packet latency of different traffic flows. The performance of the algorithm was also tested when the ts value was very small (20 times less) and very large (20 times more) than this calculated period. The test was conducted in a wireless network setup using two end devices, End device 1 and End device 3, with the physical data rate values fixed to 24 Mbps and 6 Mbps respectively as shown in Figure 5.3. End device 4 and End device 6 are used as receiver end devices. The values of $cwnd$ and instantaneous airtime consumed are used as KPIs to measure the sensitivity to timeslot length. The test was run for a fixed period and the outcome of $cwnd$ and instantaneous airtime consumption value used by the algorithm for different ts periods by End device 1 is plotted in Figure 5.4 and Figure 5.5 respectively.

For the case when ts is 20 times less, the ts period is approximately 48 ms and it is very short, hence the AP cannot consider the airtime usage of all the end devices (even in this test case with only two end devices) using the network for such a short period. Hence it appears to AP as if End device 1 is overusing the

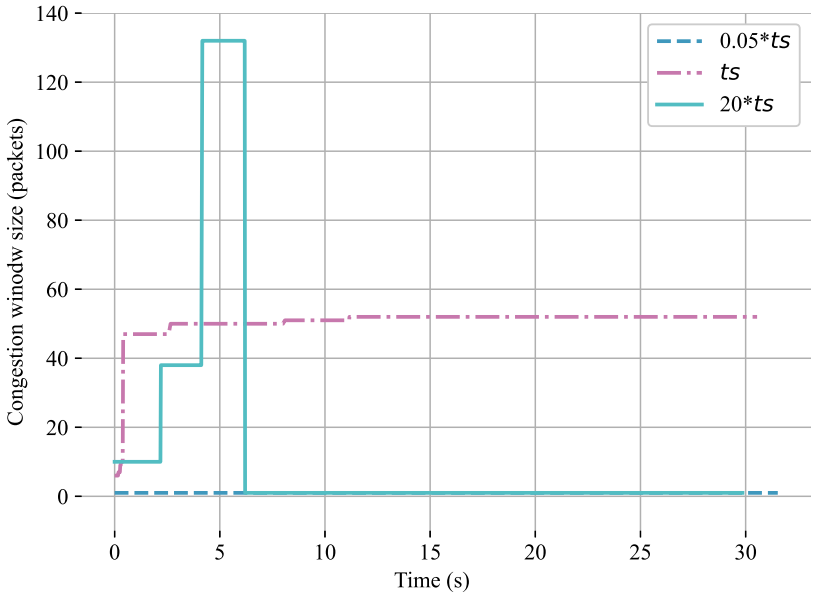


Figure 5.4: $cwnd$ for different ts at End device 1.

network. Therefore, it asks the End device 1 to reduce the airtime usage in the form of feedback with every ACK packet. The same effect can be seen in Figure 5.4 where the $cwnd$ value is kept to the minimum possible value, the impact of which can be seen in Figure 5.5 where the airtime usage is very low. Though the airtime usage is very low, because of the very small ts value, the airtime usage appears to be relatively high compared to the setpoint sp (in this case, 0.4).

For the case when ts is 20 times higher, the airtime consumption of the end device is accumulated over a large period and the feedback on airtime usage is sent relatively late. Meanwhile, if the end device exceeds or subceeds the allocated usage, it will be reduced or increased after the late feedback due to the large ts. Additionally, the $cwnd$ value is significantly decreased or increased based on the feedback and large ts value and has a negative impact over the next ts. Similarly, the cycle continues over the next timeslots and the throughput of the end device is either too high with several packet losses or too low with a very low data transfer rate. The application data transfer rate will always remain in the extremes as shown in Figure 5.4. In Figure 5.5, the inital ts value was approximately 6 s and the airtime consumption of the end device is accumulated till the 6th second and the device receives the feedback to reduce the window size extensively. This resulted in large amount of data in the network thus increasing the RTT, therefore the ts was increased to 22 s. On reducing the size, though the device underutilizes the

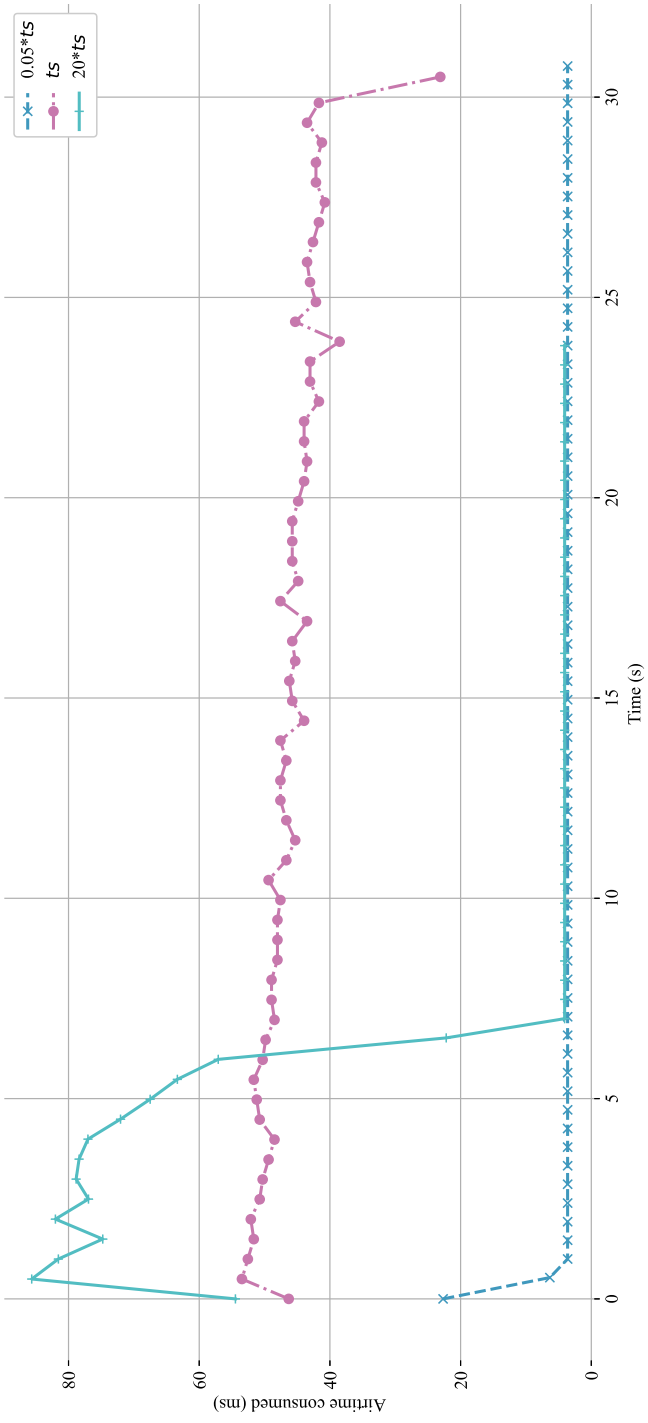


Figure 5.5: Instantaneous airtime consumption for different ts values.

timeslot; since the timeslot is very large, the feedback is not received immediately.

5.5.3 Benchmarking against traditional CC algorithm

The performance of the FCCC algorithm was benchmarked against the CUBIC CC algorithm which is traditionally used by default in all the networking devices. The KPIs used to validate the performance of FCCC are improvement in the throughput and airtime fairness between the wireless devices. To measure these values, the tests were conducted in a wireless network setup in Figure 5.3 with three end devices, End device 1, End device 2, and End device 3, with physical data rates fixed to 24 Mbps, 12 Mbps, and 6 Mbps respectively acting as sender end devices. The other end devices End device 4, End device 5, and End device 6 act as receiver end devices.

As it is essential to see the impact of the designed algorithm on the consumption of airtime, instead of real-time applications, a fixed payload of 5 MB was sent from End device 1, End device 2, and End device 3 at the same time using socket programming. This is used to evaluate the improvement in throughput. The measured average throughput values of all the end devices using FCCC and CUBIC algorithms are plotted in Figure 5.6. The percentage of average throughput improvement per end device when using the FCCC algorithm as compared to CUBIC CC is also measured and plotted in Figure 5.7. From the plots it can be noticed that all the end devices have the same throughput while using the CUBIC CC algorithm, irrespective of the physical data rates of the end devices. Whereas in the case of FCCC, the end device with a higher physical data rate has 58% better throughput as compared to the CUBIC and other end devices. The throughput of end device with the lowest physical data rate is 43% lower while using FCCC as compared to the performance while using CUBIC. Figure 5.7 also contains the performance of the RACC algorithm (designed in Chapter 4) as compared to the CUBIC. Though the performance of the end device with dr 24 Mbps is slightly higher in the case of RACC, the reduction in the performance of the device with dr 12 Mbps is very high. As discussed in Section 4.7 of Chapter 4, the end devices with lower physical data rates do not fully utilize the available channel bandwidth, as the airtime feedback from AP is explicit in nature.

From the instantaneous airtime consumption plot in Figure 5.8, it can be noticed that along with the improved throughput of end devices with higher physical data rates, FCCC also achieves airtime fairness among the end devices. This is achieved by utilizing $cwnd$ accordingly as per the FCCC algorithm, as shown in Figure 5.9. On starting the TCP connection simultaneously in all the end devices, the End device 1 has higher $cwnd$ at 0th second. Once the data transfer of End device 1 is ended at around 13th second, the $cwnd$ values of other end devices are updated accordingly. When all the end devices are active, we can see an unequal cyclic

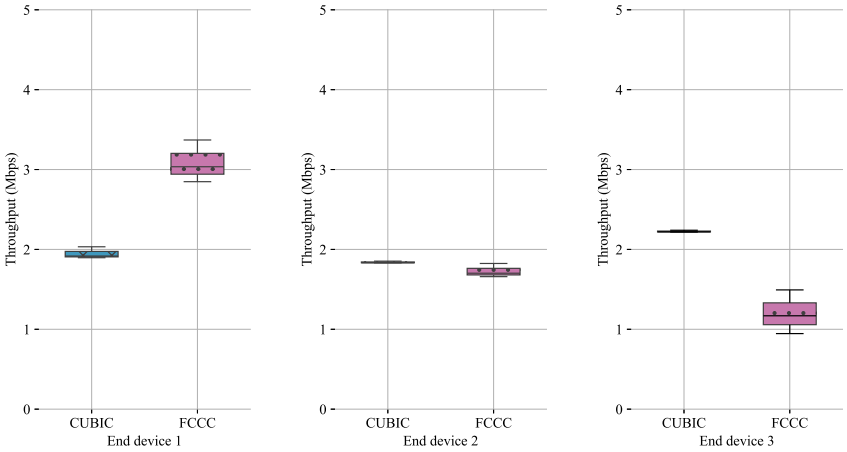


Figure 5.6: Average throughput of the FCCC algorithm as compared to the CUBIC.

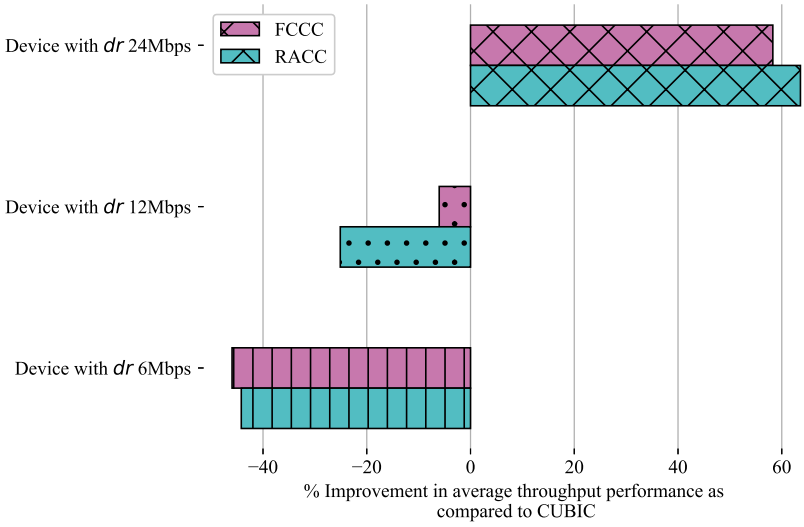


Figure 5.7: Percentage change in average throughput of the FCCC and RACC algorithms as compared to the CUBIC.

pattern in the application data transfer rate resulting in equal airtime usage.

To measure the improvement in airtime consumption of the end devices with higher physical data rates, the test was conducted where all the end devices continuously sent data for 30 seconds for FCCC and CUBIC algorithms. The percentage difference between the average airtime consumed by the end devices is plotted in Figure 5.10. Results indicate that in the case of CUBIC, the End device 1 with a lower physical data rate is dominating in airtime consumption, therefore the percentage of the average difference in airtime consumption among the end devices is higher than 100% and negative in the case of CUBIC. In contrast, the FCCC aims at airtime fairness, thus all the end devices consume equal airtime with a maximum difference of 18%. The Figure 5.10 also shows the performance of the RACC algorithm. Though the performance of the RACC algorithm is not as good as that of the FCCC algorithm, it is still better than that of the CUBIC algorithm. As the AP feedback is explicit, the algorithm RACC responds to it right away without considering the past or making predictions about the future, in a fixed ts. Therefore, there is a chance that some devices will use the wireless bandwidth excessively or insufficiently. Whereas the FCCC decides $cwnd$ in a controlled loop format considering the past and future events over a dynamically adjusted ts.

A similar effect can be seen in the instantaneous airtime consumption of the end devices plotted in Figure 5.11.

In summary, the results prove that using FCCC, the throughput of end devices with higher physical data rates is better in comparison to the throughput from using the CUBIC algorithm. This is the outcome of airtime fairness achieved by FCCC among the end devices. Though there is a small degradation in the throughput of those devices with lower physical data rates this is the penalty that end devices with low data rates must pay to achieve airtime fairness and improve the overall performance of higher physical data rates.

5.5.4 Robustness of FCCC algorithm to changing data rates

The robustness of the designed FCCC algorithm to changing physical data rates was tested in a wireless setup shown in Figure 5.12a. The physical data rate of End device 1 is switched between 24 Mbps and 6 Mbps every second. The End device 3 acts as receiver end device. The behavior of $cwnd$ and $cwnd_{up}$ for varying physical data rates are the KPIs for this test and the outcome is plotted in Figure 5.13. $cwnd_{up}$ is dependent on the data rate and acts as an upper bound to the $cwnd$. Therefore, when the physical data rate goes down at 2nd second in Figure 5.13, the $cwnd_{up}$ instantly goes down according to the physical data rate and thus the data transfer rate $cwnd$. Similar effects can be seen when the physical data rate goes up at 3rd second. The robustness of FCCC helps in maintaining airtime fairness in case of improvement or deterioration in the physical data rate

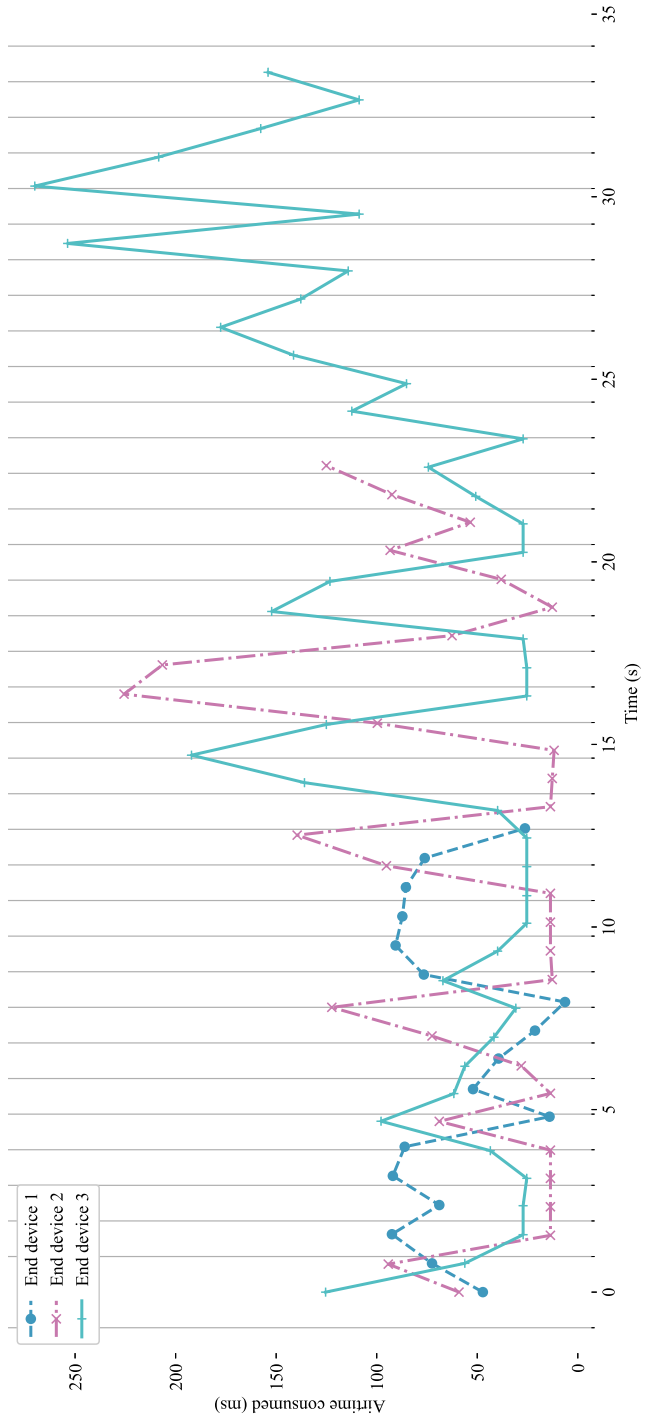


Figure 5.8: Instantaneous airtime consumption of the FCCC algorithm on sending a fixed data payload of 5 MB (minor grid lines indicate ts).

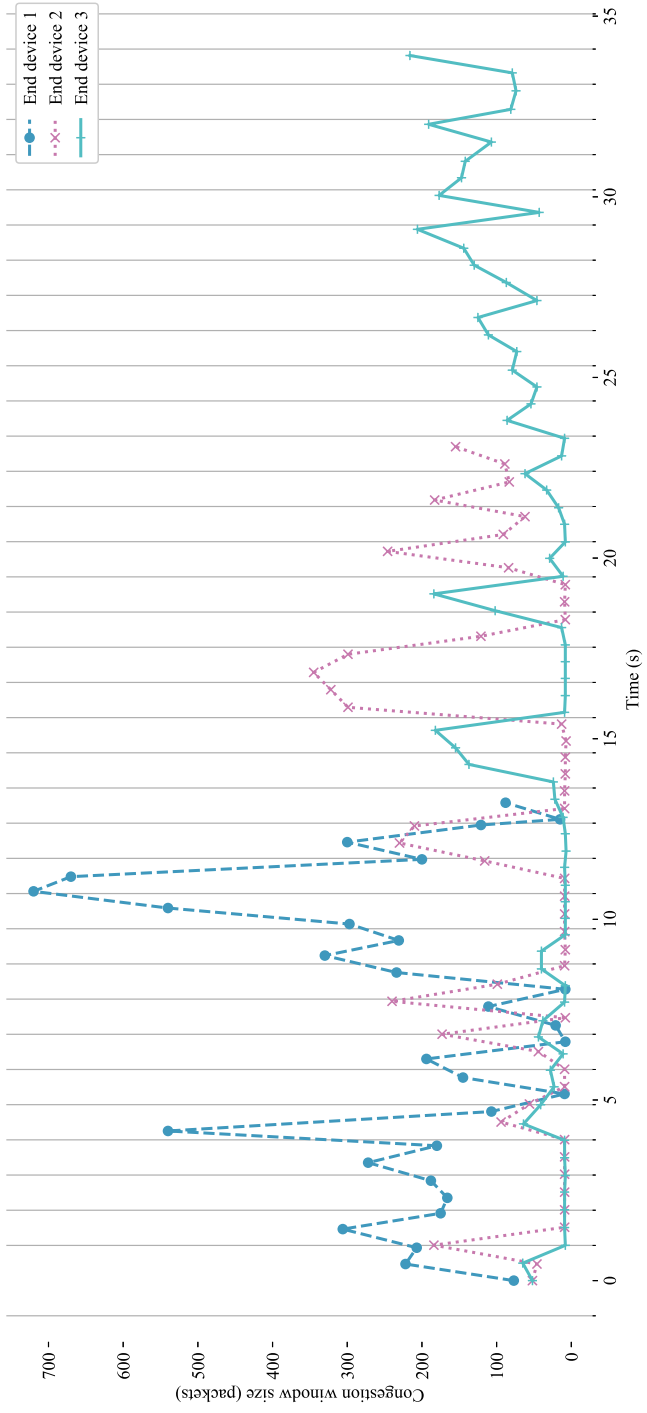


Figure 5.9: Behaviour of $cwnd$ of the FCCC algorithm on sending a fixed data payload of 5 MB (minor grid lines indicate ts).

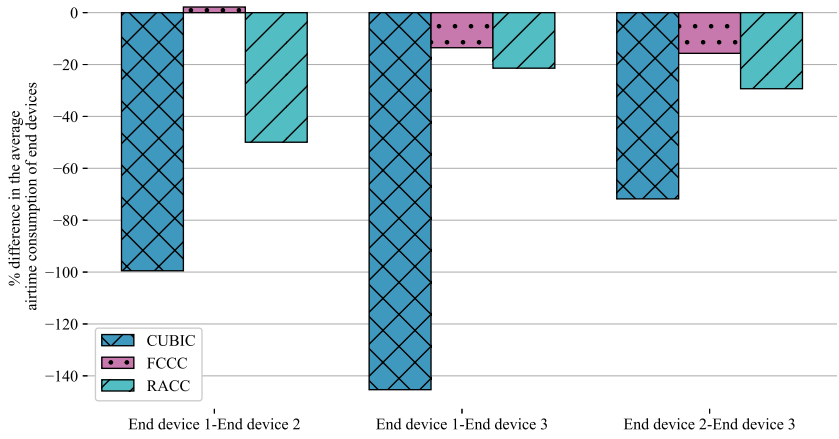


Figure 5.10: Percentage difference between average airtime consumption of end devices.

and thus avoiding the biased decision at the AP among the end devices.

5.5.5 Use cases: different sp values from AP

The sp value shared by AP has a significant impact on deciding the application data transfer. Varying the sp values for different end devices adds additional functionality of priority-based airtime fairness to the designed network architecture. Tests were conducted in the wireless network setup in Figure 5.12b, while sending a fixed amount of data from each end device, to see the behavior of application data transfer rates and the airtime consumption when different sp values are sent as feedback to different end devices. End device 1 has fixed physical data rate of 24 Mbps and sp value of 0.45 and the End device 2 has a fixed physical data rate of 24 Mbps and sp value of 0.15. End device 3 and End device 4 act as receiver end devices. The measured instantaneous airtime consumption and the behavior of application data transfer rate i.e., $cwnd$ being the KPIs, are plotted in Figure 5.14 and Figure 5.15 respectively. Results indicate that for the sp value of 0.45, the $cwnd$ value is higher, and thus the airtime consumption of End device 1. The total airtime consumption of the end device with sp value 0.45 is three times greater than that of the one with sp value 0.15. Unlike the previous cases, after the end of the data transfer of End device 1, the $cwnd$ value of the other end device doesn't increase significantly, and the device keeps the airtime consumption as low as possible because of the feedback of 0.15 sp value from AP. This experiment has proven that priority-based airtime fairness can be implemented by AP by varying the sp values. This opens the door for use in multiple application scenarios.

Even with the increase in the number of nodes, each end device will get a

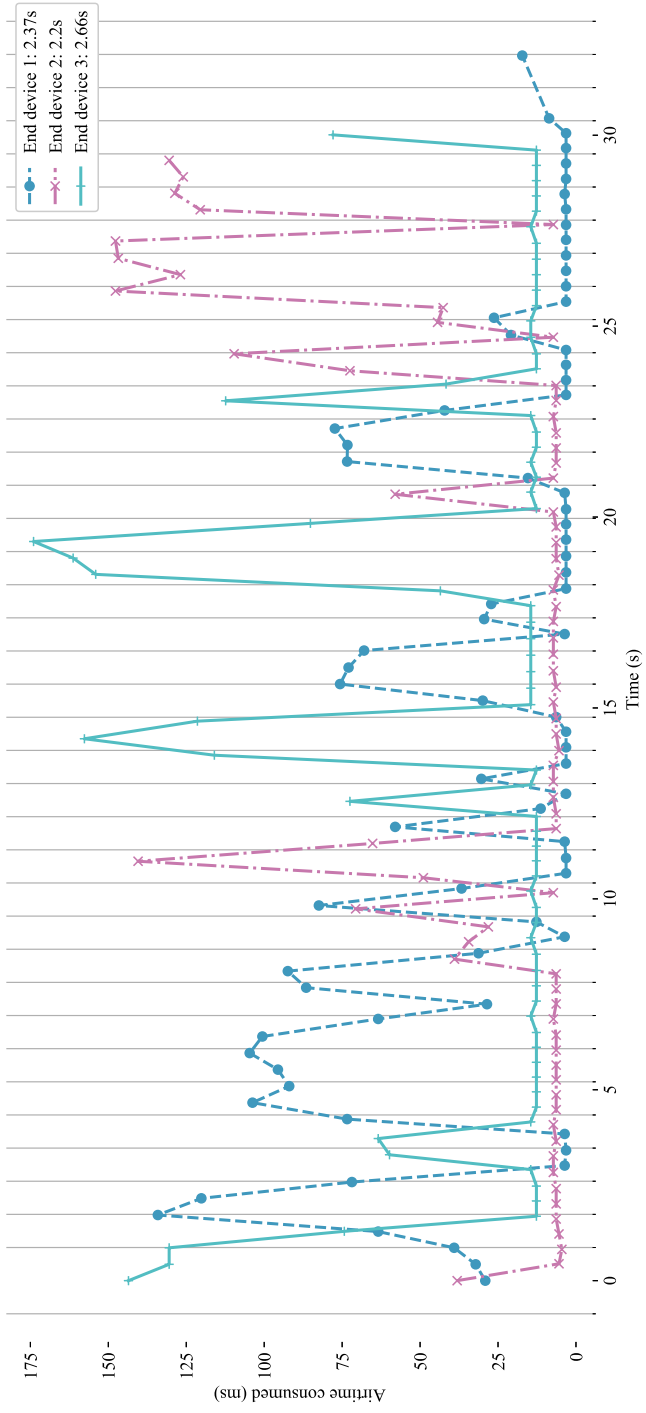
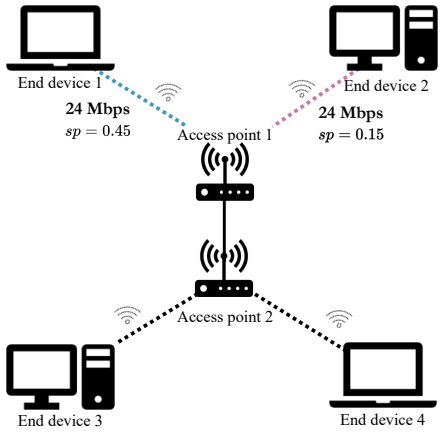


Figure 5.11: Instantaneous airtime consumption of the FCCC algorithm on sending data for 30 s (minor grid lines indicate ts).

(a) Wireless network setup to validate the robustness of the FCCC algorithm (Section 5.5.4).



(b) Wireless network setup to test the use case of varying sp value (Section 5.5.5).

Figure 5.12: Wireless network setup for test cases in Section 5.5.4 and Section 5.5.5

certain amount of airtime allocated by the AP, and in the worst-case scenario, the algorithm is designed to send at least $cwnd_{init}$ number of packets to sustain the connection and avoid starvation. Also, AP monitors the airtime consumption of each end device and hence the end device that is already registered but has not been sending packets in each timeslot will be given an opportunity and the other end devices will be notified to reduce the application data transfer rate to balance the airtime fairness. The transport layer buffers are designed to be large enough to cater to the TCP needs such as re-transmissions and adjusting to the receiver buffer size. Hence no significant buffer overflow situations are noticed in transport layer buffers. Since the number of packets reaching the MAC layer of the device is monitored by the transport layer, the MAC layer buffer does not experience excessive packet queuing or buffer overflow.

5.6 Conclusion

Wi-Fi is an essential tool for connectivity in today's and future private-professional networks, offering various data rates based on channel quality. Despite Wi-Fi

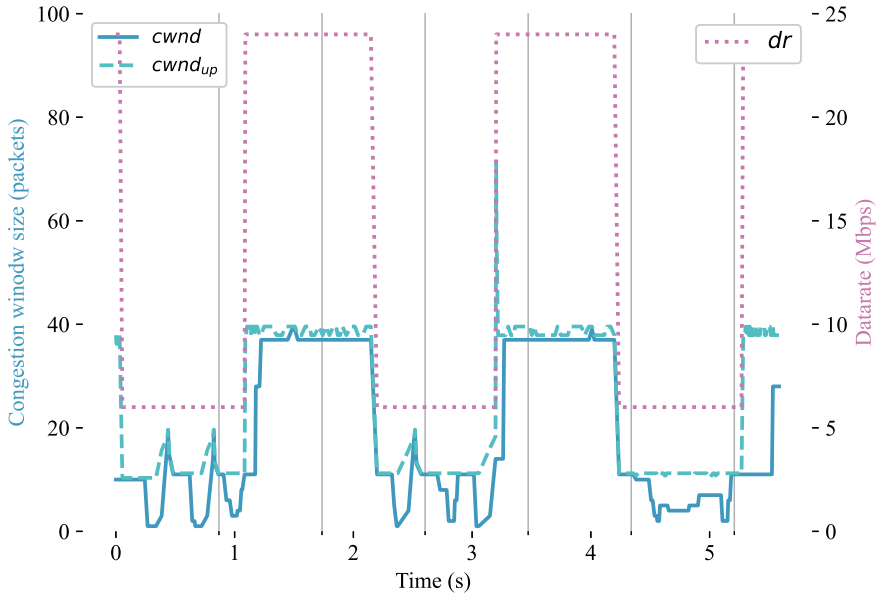


Figure 5.13: The $cwnd_{up}$ and $cwnd$ for changing physical data rates (minor grid lines indicate ts).

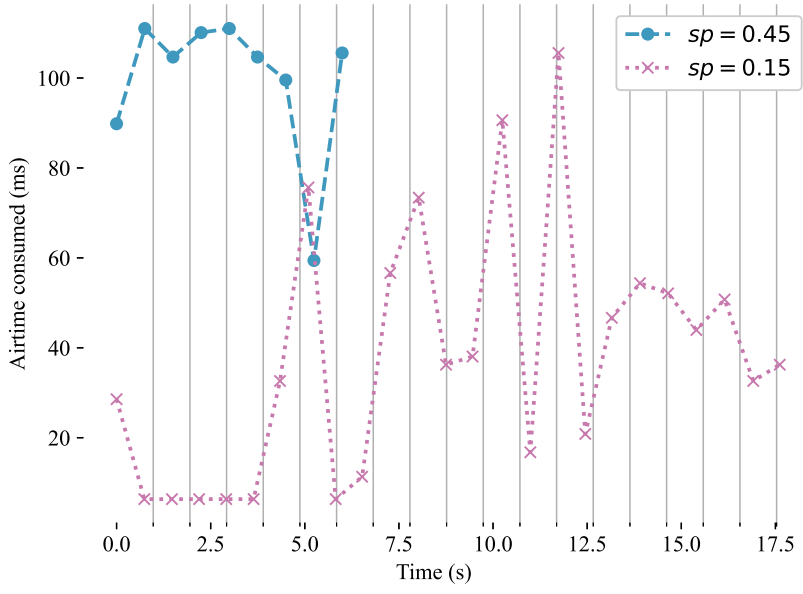


Figure 5.14: Instantaneous airtime consumption for different sp values (minor grid lines indicate ts).

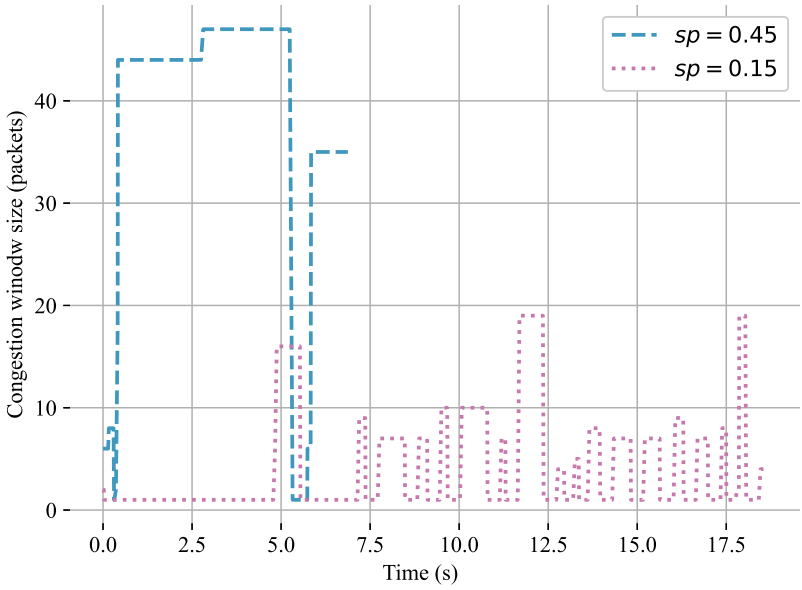


Figure 5.15: Results of cwnd for different sp values from AP.

protocol advancements, throughput degradation remains a problem for higher data rate devices when lower data rate devices are present, due to airtime unfairness. Emerging applications with diverse requirements highlight the need for network protocols serving these demands. Innovations like INT, APP-NET, and eBPF allow real-time monitoring, but CC algorithms, designed for wired networks, need adaptation for efficient data transfer in wireless networks. Therefore, there is a need for a suitable algorithm to determine optimal data transfer rates without sacrificing throughput in the presence of lower data rate devices in wireless networks. In this work, we address this problem by designing a feedback-based control-loop CC algorithm, FCCC. The system implements additional bookkeeping of the airtime consumption of each end device in AP, which is further added to the INT header as aggregated airtime feedback to each end device. The FCCC algorithm considers the real-time aggregated airtime feedback from AP and network state information obtained through INT to decide the suitable $cwnd$. The designed algorithm was benchmarked against the existing CUBIC CC algorithm, whereby using the FCCC the end device with a higher physical data rate had a 58% improvement in the throughput as compared to using the CUBIC algorithm. The airtime difference between the end devices was as low as 18%. The performance of the FCCC algorithm was also compared against the RACC algorithm designed in Chapter 4. Though both the algorithms aim at achieving airtime fairness, since FCCC is based

on a PI controller, unlike RACC, it achieves better airtime fairness. The algorithm was validated for its sensitivity to the varying timeslot length, and robustness to the changing physical data rate. A potential use case of priority-based airtime fairness was also tested for future use cases.

The NACC algorithm, which was designed in Chapter 3, focuses on listening to the demands of the application and meeting those needs. The technique is resilient to the abrupt changes in wireless network environments since it also takes the current network condition into account when determining the $cwnd$. Whereas, FCCC is designed to achieve airtime fairness like RACC, but in a proactive manner.

The implemented FCCC algorithm improves the throughput of the higher physical data rate devices in the presence of lower physical data rates in wireless networks. The additional features in the algorithm and the monitoring and feedback system in AP, enable priority-based airtime fairness in wireless networks and are an outlet for multiple such use cases that can be explored in the future.

References

[1] 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 Transactions on Network and Service Management, 18(1):627–641, 2020.

[2] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke. *Tighter application-network interfacing to drive innovation in networked systems*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 53–57, 2021.

[3] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke. *Adaptive transport layer protocols using in-band network telemetry and eBPF*. In 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pages 241–246. IEEE, 2021.

[4] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. *Restructuring endpoint congestion control*. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pages 30–43, 2018.

[5] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *CUBIC for fast long-distance networks*. Technical report, 2018.

[6] W. Eddy. *Rfc 9293: Transmission control protocol (tcp)*, 2022.

[7] J. Haxhibeqiri, R. V. Bhat, I. Moerman, and J. H. IDLab. *Age-of-Information Aware In-band Network Telemetry for Better Network Predictability*. In 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pages 42–47. IEEE, 2021.

6

Multi-Context aware RL approach towards INT-based Congestion Control Algorithm

It is advantageous to be able to predict the future network, even though the designed algorithms are flexible enough to adjust to the conditions of the wireless network. Although the PI control-loop-based CC algorithm is proactive by nature, in this chapter, a reinforcement learning-based approach was looked into in order to expand its capabilities to meet application needs and research learning methodologies. Multi-context-aware RL-based CC is modeled as a Contextual Markov Decision Process since the wireless network parameters, like channel quality, other competing nodes, and number of flows, are independent of the actions taken by an end device and have a significant impact on the overall throughput and latency of the end device. The number of application data packets transmitted to the network is adjusted based on application requirements and real-time network information via three different multi-context-aware RL-based CC algorithms that are presented. In the Wi-Fi network, two of these solutions were put into practice and tested.

⋆ ⋆ ⋆

This chapter is based on:
Multi-Context aware RL approach towards INT-based Congestion Control Algorithm

Accepted in IEEE Conference on Standards for Communications and Networking, November 2024.

⋆⋆⋆

6.1 Introduction

Private wireless networks are increasingly adopted across sectors like industry automation, healthcare, and military due to their enhanced coverage, performance, security, and adaptability to AI and IoT technologies. These networks must meet the high demands of applications like AR/VR and cloud computing in terms of throughput, latency, reliability, and application prioritization. Though techniques like slicing, priority queuing, and DiffServ are offered by the intermediate network nodes, the massive increase in traffic (number of flows in the range of 100s–1000s) with different priority requirements ultimately exceeds their capabilities on incorporating new functionalities. TCP is widely used as end-to-end transport protocol for its reliable, error-free data transfer and CC. However, most CC algorithms are designed for wired networks, overlooking wireless-specific issues like interference, mobility, and lossy mediums. As network traffic grows, ensuring congestion-free services alongside reliability and service differentiation is critical in wireless networks.

The transition to SDN, which separates control and data planes, allows for full programmability of the forwarding plane [1]. To maximize this, understanding application requirements and real-time network context is crucial. INT techniques, superior to out-of-band methods, have been adapted for wireless networks [2]. An APP-NET interface designed in [3] allows applications to specify their traffic requirements to the network layer and understand the real-time network context. Design goals for the future private wireless networks (6G and Wi-Fi)[1] involve integration of INT and APP-NET in making decisions. The end devices connecting to private networks are also now more powerful in terms of memory, computation, and programmability than before. Innovation in Linux kernels like eBPF has enabled the programmability of network protocol stacks in the end devices [4].

Over time, wireless technology adoption in private-professional networks has grown, as has the variety of applications available in these networks. Future wireless networks will therefore be extremely complicated, making traditional mathematical approaches to network design, implementation, and operation inadequate. Dynamic programming and heuristics are also suitable to optimize the $cwnd$ to maximize throughput and reduce latency. However, dynamic programming breaks down the problem in steps and solves it recursively. It is time-consuming and adds additional overhead to take the decisions in real-time, especially in the case of

[1] Unified Networking Experience defined by Nokia Bell Labs: https://www.bell-labs.com/research-innovation/projects-and-initiatives/unext/

wireless networks. Also, the wireless network is highly unpredictable and dynamic, and dynamic programming is suitable for more stable conditions where the results can be reused. The above-mentioned reasons have motivated a move towards data-driven algorithms (elaborated in [5]).

Leveraging the computational capabilities of the end devices and data programmability of the forwarding plane along with INT and APP-NET integration, this chapter proposes a solution where the end devices collaborate with the intermediate network nodes in achieving optimal network operation. As a first step, a novel multi-context-aware RL-based CC algorithm is introduced that adapts the number of application data packets sent to the network based on application requirements and real-time network information. Since the wireless network parameters such as channel quality, other competing nodes, and number of flows, being independent of the actions taken by an end device, have a significant influence on the overall throughput and latency of the end device, the solutions proposed in this chapter are based on multi-context-aware RL-based CC that is modeled as a Contextual Markov Decision Process (CMDP).

The key research contributions of this study are as follows:

- Model reinforcement learning-based CC algorithm for wireless networks as contextual Markov decision process.
- Design of three different multi-context-aware reinforcement learning-based CC algorithm for private wireless networks.
- Implement, train and test two of the proposed three methods using Epsilon-Greedy Q learning in w-iLab.2 Testbed.

The rest of this chapter is structured as follows: Section 6.2 focuses on related works in the area of RL-based CC algorithms. Section 6.3 elaborates on the challenges and Section 6.4 explains the proposed solutions for implementing an RL-based solution for wireless networks. Section 6.5 gives an overview of preliminary results using Q-learning. Section 6.6 concludes the chapter with insights on the future work.

6.2 Related work

This section discusses research works that focus on reinforcement learning-based CC algorithms. The survey paper [6] gives a detailed description of different RL techniques used to implement CC algorithms for different network scenarios. Some of the commonly used value-based techniques are Q-learning and deep Q-learning, whereas policy-gradient, Actor-Critic (AC), Advantage Actor-Critic (A2C), and Asynchronous Advantage Actor-Critic (A3C) are commonly used value-based learning techniques. Based on the survey, most of the implementations update

$cwnd$ for specific scenarios. A few of the algorithms also manage the queue length based on the current state obtained from congestion notification. In such scenarios, Proportional Integral Derivative (PID) is generally used in the RL technique. Despite showing strong learning capabilities, the current RL-based CC algorithms do not converge easily and find state abstraction challenging. In addition to that, the algorithms require a huge amount of storage for states and actions and have high computational complexity.

In [7], the authors have proposed an adaptive CC algorithm for wireless networks based on deep-RL, where the algorithm classifies the flow using deep neural networks and finds the appropriate $cwnd$ using a global repository based on the flow classification. The solution is implemented and tested in a simulator. The authors of [8] have proposed TCP-Gvegas, an improved TCP Vegas based on grey prediction theory for multi-hop ad hoc networks. The algorithm uses an optimal exploration method based on Q-learning and RTT quantizer. In [9], authors have proposed an RL-based CC algorithm for vehicular communication where $cwnd$ is chosen based on the current channel conditions.

Unlike the above-mentioned works that use partial information obtained from TCP ACKs as state information, the algorithm proposed in this chapter utilizes real-time network information obtained from INT as state information. This rich INT information also helps to reduce the state space. Additionally, the above mentioned works either fail to consider the wireless channel conditions or use very few wireless parameters as state information. The proposed multi-context-aware RL-based algorithm considers wireless channel conditions as contexts and is based on CMDP. Moreover, the algorithm proposed in this chapter is trained online in the w-iLab.2 testbed on wireless devices that are available commercially.

6.3 Challenges

One of the main challenges is the ever-changing dynamics and unpredictability of wireless networks. Due to the presence of several dynamic network parameters, like channel conditions, intermediate network node context, and competing nodes in private wireless networks, it is challenging to confine the state and action space. With the addition of diverse requirements for applications, the state and action space increases exponentially. In addition to that, it is essential to achieve short-term performance enhancement (such as throughput, latency) along with the long-term goal (congestion-free, service differentiation). Since service differentiation and adaptability are the core features of private networks, it is crucial to address these challenges. The challenges involved in designing an RL-based solution specific to Q-learning-based algorithms for wireless networks are outlined in [10]. Some of the important challenges involve adaptation of Q-table to abruptly changing environment, effects of exploration on the stability of the algorithm, and achieving

stable Q-values. The other issues addressed are in the area of multi-agent RL algorithms.

6.4 Proposed solution

6.4.1 RL model

The existing CC algorithms decide the $cwnd$ based on the partial information available from ACK packets. Keeping in mind the goals of future private wireless networks, it is essential to integrate INT and APP-NET. Leveraging this, an RL-based CC algorithm that utilizes real-time network information from INT is designed. Some of the INT monitoring parameters that are relevant for CC algorithms are buffer capacity, number of packets arriving, number of flows in intermediate nodes, airtime usage, and physical data rate of each end device [11], [12]. Parameters like buffer capacity, airtime usage are directly affected by the chosen $cwnd$ value, the number of flows in the intermediate node and the physical data rate are independent of the decision made by the CC algorithm. Hence they act as context for which the RL algorithm must be trained.

The unpredictability and dynamic of wireless networks further make it challenging. The existing RL-based algorithms are trained based on partial network information and do not consider the changing dynamics of wireless networks as context. Having a detailed insight into the network would also reduce the number of state and context spaces used to train the algorithm. Therefore the RL-based CC algorithm is modeled as CMDP (C, S, A, $M(c)$) where C is called the context space (independent of the actions), S and A are the state and action space respectively. M is the mapping function of context $c \in C$ to an MDP $M(c) = (S, A, p_c(s'|s,a), r_c(s'), \pi_{c0})$, where $p_c(s'|s,a)$ is the transition probability from state s to s' ($s', s \in S, a \in A$), $r_c(s')$ is the reward obtained when moving from state s to s', and π_{c0} is the initial state distribution for context c. Since the end device is aware of the real-time network information through INT, we consider that the context space is observable.

6.4.2 Proposed solutions

Using the real-time network information available through INT significantly reduces the state space, as the algorithm now has a detailed insight into the ongoing changes. In the previous implementations of CC algorithms, it has been proven that using INT information gives better view of the network resulting in simpler and better performing CC algorithms [11], [12]. Keeping in mind the challenges addressed in Section 6.3, three multi-context-aware RL-based CC algorithm solutions are proposed as follows:

1. **Fixed contexts**: This is inspired by existing recommendation systems where individual contexts are considered, and since the context space in wireless networks is observable through INT, the RL-model is trained separately for each of these contexts as shown in Figure 6.1. Though the algorithm addresses the issue of unpredictability of wireless networks by training the models for different contexts, it comes with the disadvantage of large context space. Additionally, it is a tedious task to emulate all the possible contexts, especially for a wireless network, to train the model. If we take the instance of number of flows in the network, different models must be trained for different scenarios of number of flows in the network.

2. **Fixed number of relative contexts**: Instead of considering the exact value of the context, the algorithm is trained for relative context values. The algorithm starts with the stable context level; whenever there is a change in the context, for instance, an increase or decrease in the number of flows or the physical data rate, then it moves to a different context level. The solution drastically reduces the context space and can be easily adapted to any wireless network as it considers only the relative values of the context. The convergence of the algorithm might be longer compared to the first case as each stable context level is different in each scenario.

   Figure 6.1 summarizes the proposed solutions where each color indicates particular context for example, number of traffic flows and each color gradient layer of this color indicates different physical data rates supported by the channel such 6 Mbps, 12 Mbps, etc. These separate layers are compressed in relative contexts. The resulting number of context values for fixed contexts solution is more than that of relative contexts solution as indicated in Figure 6.2.

3. **Dynamic number of contexts**: A cloud-based solution is proposed where the end device can dynamically add the context layers and partially retrain on top of the existing trained model obtained from the cloud. The cloud maintains a set of trained models. Whenever the end device requests a trained model for a certain context, the cloud defines to what extent the trained model is different from the requested model and finds the model that is closest to the requested context. The end device can then use this model with a specific exploration rate to customize it to the current context. The exploration rate is dependent on the relative difference between the current context and the context of the trained model from the cloud. The end device posts this locally trained model back to the cloud. Along with confining the context space and addressing the unpredictability of the wireless networks, the solution also maintains a homogeneous model and reduces the computational and training expense at each end device. The network architecture used for such a system

Table 6.1: The number of context combinations and training complexity for the proposed multi-context-aware RL CC solutions.

Solution	Number of context combinations	Training complexity
Fixed contexts	$n = k_1 * k_2 * ... * k_x$	$\mathcal{O}(n)$
Relative contexts	3^x	$\mathcal{O}(3^x)$
Dynamic contexts	$n = k_1 * k_2 * ... * k_x$	$\mathcal{O}(\frac{n}{m})$

is represented in Figure 6.3.

The number of context combinations and training complexity for each of these proposed solutions is listed in the Table 6.1. In fixed contexts case, for x distinct contexts, with k_x possible values for each context, the total number of context combinations is $n = k_1 * k_2 * ... * k_x$. Therefore, the training complexity is equal to $\mathcal{O}(n)$. In the case of relative contexts, for x distinct contexts, there are three possible values for each context. Hence the number of context combinations is 3^x and the training complexity is $\mathcal{O}(3^x)$. The training complexity of the relative context solution is less than the previous solution. For the solution of dynamic contexts, for x distinct contexts, with k_x possible values for each context, assuming the end device receives a trained model from the cloud for m context combinations, the training complexity is now reduced to $\mathcal{O}(\frac{n}{m})$.

6.5 Q-learning approach for contextual-MDP

As a preliminary test, Epsilon-Greedy Q-learning is used to model multi-context-aware CC algorithm. The Q value equation for CMDP can be formulated as: $Q : (C, S)\text{x}A \rightarrow R$

$$\begin{aligned} Q'_c[s_t, a_t] \leftarrow Q_c[s_t, a_t] + \alpha(r_c(s_{t+1}) + \\ \gamma(max_a(Q_c[s_{t+1}]) - Q_c[s_t, a_t])) \end{aligned}$$

The equation can be further elaborated as,

$$\begin{aligned} Q'[c_t, s_t, a_t] \leftarrow Q[c_t, s_t, a_t] + \alpha(r_c(s_{t+1}) + \\ \gamma(max_a(Q[c_{t+1}, s_{t+1}]) - Q[c_t, s_t, a_t])) \end{aligned} \tag{6.1}$$

where α and γ are the learning rate and discount factor respectively [13]. $Q'_c[s_t, a_t]$ and $Q_c[s_t, a_t]$ are the new and current Q values respectively. $r_c(s_{t+1})$ is the reward for moving from the state s_t to s_{t+1} and $max_a(Q_c[s_{t+1}])$ is the maximum expected future reward obtained from context c_{t+1} and state s_{t+1}. c_{t+1} and s_{t+1} are the context and state in the next time slot ($next\_ts\_context$, $next\_ts\_state$). s_{t+1} is the result of taking action a_t. c_{t+1} is independent of a_t and can be same or different

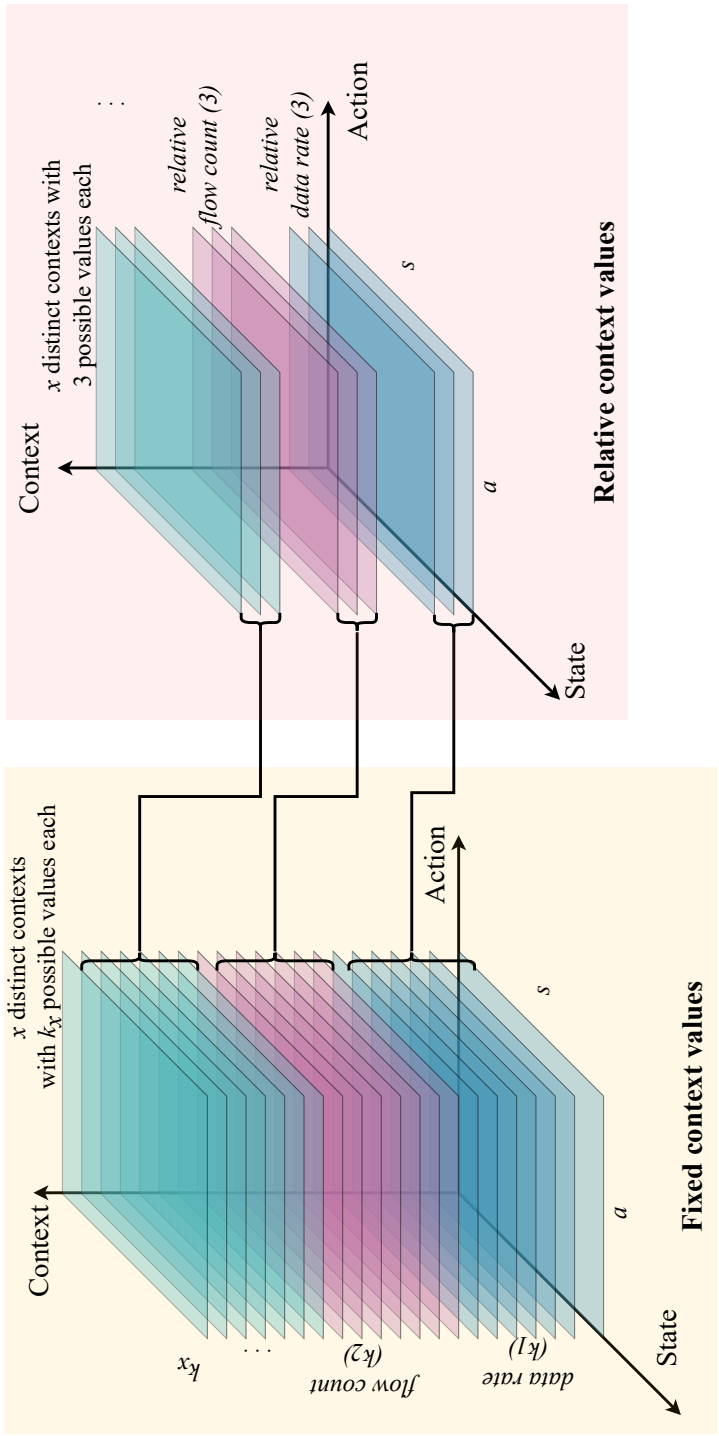


Figure 6.1: The figure indicates the compression of contexts to relative contexts which in turn reduces the total number of context values while training as shown in Figure 6.2.

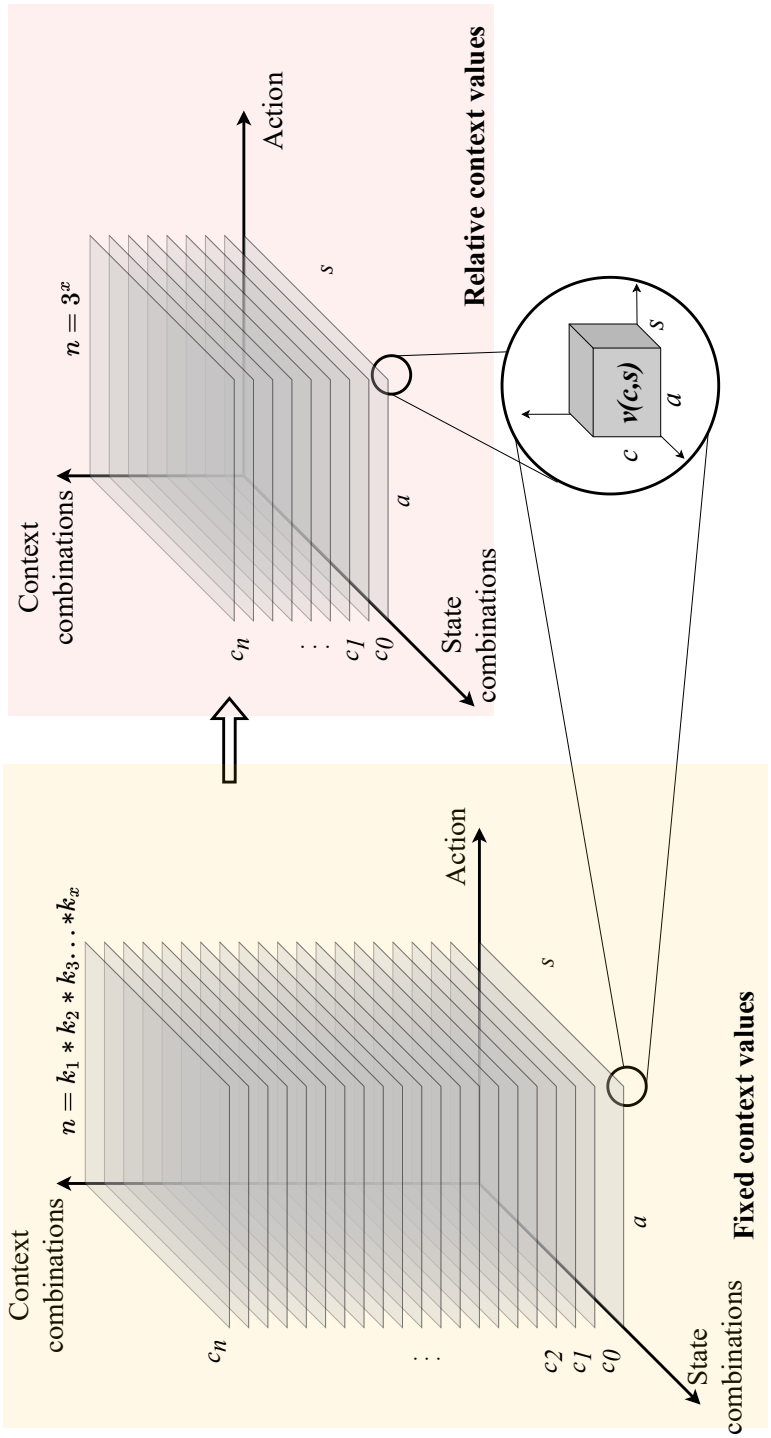


Figure 6.2: $v(c, s)$ is the maximum expected future reward (Q value) obtained on taking action a at context c and state s.

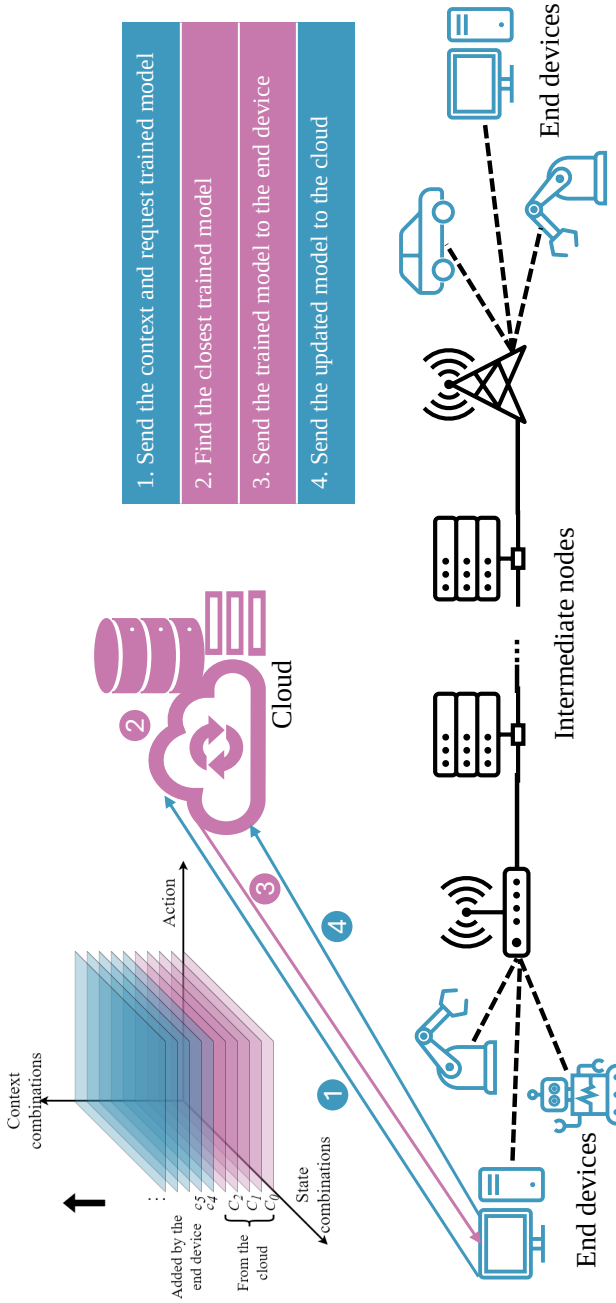


Figure 6.3: Cloud-based solution to train multi-context-aware RL-based CC algorithm

from the previous context. In case of change in context, $max_a(Q_c[s_{t+1}])$ is chosen from the Q-table of the changed context and uses this table until there is change in the context.

6.5.1 Results: Fixed context space

To evaluate the feasibility of the proposed solutions, CC is designed for a Wi-Fi network with fixed context (as per system 1 in Section 6.4.2). The context, state and action spaces are defined below:

- **Context**: for a fixed number of flows in the network at all the times, the physical data rates are fixed to 6 Mbps, 12 Mbps, 24 Mbps.
- **State**: change in the packet arrival rate at the intermediate nodes (here AP) measured in four levels, airtime usage of the end device over a fixed timeslot, measured in four levels.
- **Action**: Four action levels are defined, as follows:
  - 0→$cwnd = cwnd + 1$
  - 1→reduce the $cwnd$ by the number of packets in flight
  - 2→increase the $cwnd$ by the number of packets ACKed (ACK_p)
  - 3→$cwnd = cwnd$
- **Reward**:

$$100 * \frac{ACK_p - loss_p}{cwnd}$$

  where $loss_p$ is the number of packets lost for $cwnd$ packets sent.
- **Q table dimension**: $context \times state \times action$: $3 \times 16 \times 4$
- $\alpha \leftarrow$0.8 and $\gamma \leftarrow$0.9
- $\epsilon \leftarrow$ 1 for training and 0 for testing

The algorithm represented in Algorithm 4, was implemented in Python 3.8 programming language using CCP framework of MIT [14] on commercially available wireless devices. The algorithm was trained online for different fixed contexts for more than 20,000 steps for each context in the w-iLab.2 testbed on the ZOTAC wireless node[2]. The node operates on Ubuntu 18.04 operating system, with kernel version 4.15.0-45-generic. Since the model is trained online and doesn't require retraining. But the trained algorithm is limited for certain contexts.

[2]w-iLab.2 hardware details: https://doc.ilabt.imec.be/ilabt/wilab/hardware.html#w-ilab-2-hardware

Algorithm 4 Q-learning based Context-aware RL (C-RL) CC algorithm with fixed contexts

1: $init\_cwnd \leftarrow 10$
2: TCP (bytes ACKed, packet loss, etc.) and INT report (packet arrival rate, no. of flows, etc.)
3: Set $next\_ts\_context$ and $next\_ts\_state$ from the report
4: Update Q value using Equation 6.1
5: Choose $action$ based on ϵ-greedy method
6: Set $cwnd$ based on the chosen $action$ for next data transfer
7: Go to step 2 to receive TCP and INT report

The trained algorithm was tested on two ZOTAC nodes connected to AP (a DSS node, configured as AP, using Wi-Fi 802.11n) as two end devices, End device 1 and End device 2 with different physical data rates, 24 Mbps and 6 Mbps. The end devices sent a 10 MB application payload. As shown in Figure 6.4, using the CUBIC algorithm, both End device 1 (indicated as CUBIC 1 in Figure 6.4) and End device 2 (indicated as CUBIC 2 in Figure 6.4) take the same amount of time to transmit the data, irrespective of their physical data rates. However, using the C-RL algorithm, the End device 1 (indicated as C-RL 1 in Figure 6.4) with the higher physical data rate, has higher $cwnd$, hence completing the transfer faster than the End device 2 (indicated as C-RL 2 in Figure 6.4). This difference is also evident in the overall throughput (mentioned in the legend of Figure 6.4) for the data transfer, where the End device 1 achieved better performance with the C-RL algorithm. This also results in a better overall throughput of End device 1 using C-RL as compared to the overall throughput using CUBIC algorithm. This comes with a penalty of lower overall throughput of End device 2 using C-RL as compare to using CUBIC. This proves that the existing CC algorithms are insensitive to contexts like channel quality and competing nodes in the network, which have significant influence on the performance of end device applications. By considering these contexts while designing the RL-based CC algorithms, we can significantly improve the performance of the applications by adapting to wireless network conditions.

6.5.2 Results: Relative context space

In this section, CC is designed for a Wi-Fi network with relative context (as per system 2 in Section 6.4.2). The context, state and action spaces are defined below:

- **Context**: for a fixed data rate of 24 Mbps, the number of flows in the network is varied. The contexts of variation in the number of flows in the network are defined by three context levels: **decrease** (0), **stable** (1), and **increase** (2). **stable** context is the number of flows in the network when the end device starts its connection. It can be 2, 4, 10, 20, etc. **decrease** is the context

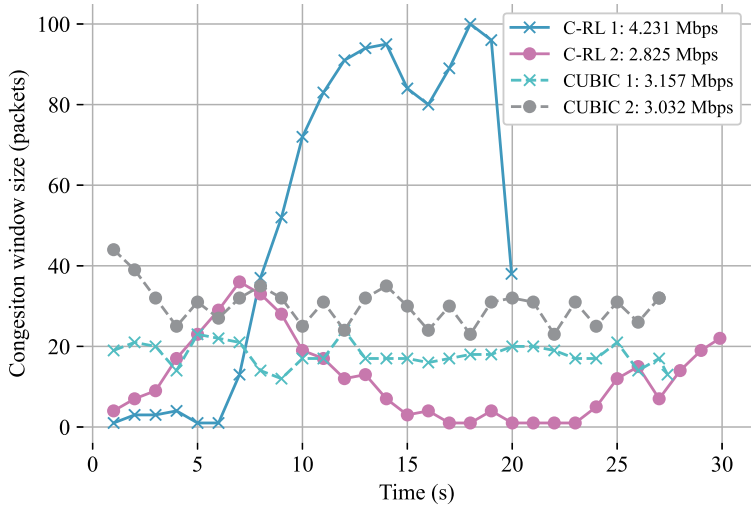


Figure 6.4: Behaviour of $cwnd$ for C-RL and CUBIC CC algorithms for End device 1 and End device 2 with physical data rates 24 Mbps and 6 Mbps respectively. (Average throughput is in the legend.)

when the number of flows in the network decreases as compared to the **stable** context; similarly, **increase** is the increase in the number of flows in the network as compared to the **stable** context. The algorithm remains in the **increase** or **decrease** context for up to ten timeslots (each timeslot is equivalent to RTT) and then moves to the **stable** context.

- **State**: change in the packet arrival rate at the intermediate nodes (here AP) measured in four levels, airtime usage of the end device over a fixed timeslot, measured in four levels.
- **Action**: Four action levels are defined, as follows:
  - 0→$cwnd = cwnd + 1$
  - 1→reduce the $cwnd$ by the number of packets in flight
  - 2→increase the $cwnd$ by the number of packets ACKed ($ACK_{packets}$)
  - 3→$cwnd = cwnd$
- **Reward**:

$$100 * \frac{ACK_p - loss_p}{cwnd}$$

where $loss_p$ is the number of packets lost for $cwnd$ packets sent.

- **Q table dimension**: $context \times state \times action$: $3 \times 16 \times 4$
- $\alpha \leftarrow$0.8 and $\gamma \leftarrow$0.9
- $\epsilon \leftarrow 1$ for training and 0 for testing
- **Training method**: The algorithm is trained for **stable** context with $\epsilon \leftarrow 1$. Once the **stable** context is sufficiently trained, the algorithm is then exposed to varying flow conditions in the network. The Q-table of state context is unchanged and the **increase** and **decrease** contexts are trained with $\epsilon \leftarrow 1$.
- **Testing method**: The algorithm is tested for **stable** context with $\epsilon \leftarrow 0$. Since the **increase** and **decrease** contexts are not sufficiently trained, these contexts are tested with $\epsilon \leftarrow 0.6$ with a decay factor of 0.1 until the value 0.2 is reached.

Using the CCP framework of MIT [14], the algorithm described in Algorithm 5 was implemented in Python 3.8 on commercially available wireless devices. The algorithm underwent online training for over 20,000 steps in several relative contexts within the w-iLab.2 testbed on the ZOTAC wireless node[3]. The node operates on Ubuntu 18.04 operating system, with kernel version 4.15.0-45-generic. Since the model is trained online, it doesn't require retraining. But the trained algorithm is limited for certain contexts. This issue is addressed by proposing the cloud-based method where the algorithm is trained for a certain number of contexts, is stored in the cloud, and is retrieved whenever necessary.

Algorithm 5 Q-learning based C-RL CC algorithm with relative contexts for number of flows in the network

1: $init\_cwnd \leftarrow 10$
2: TCP (bytes ACKed, packet loss, etc.) and INT report (packet arrival rate, no. of flows, etc.)
3: Set $next\_ts\_context$ based on relative number of the flows and $next\_ts\_state$ from the report
4: Update Q value using Equation 6.1
5: Choose $action$ based on ϵ-greedy method
6: Set $cwnd$ based on the chosen $action$ for next data transfer
7: Go to step 2 to receive TCP and INT report

From Figure 6.5 it is evident that for the **stable** context, the probability of occurrence of action 2 is higher, as the algorithm tries to increase the window size as much as possible to achieve better throughput. For the relative **increase** context, shown in Figure 6.7, contrary to the **stable** context, the algorithm prefers to either

[3]w-iLab.2 hardware details: https://doc.ilabt.imec.be/ilabt/wilab/hardware.html#w-ilab-2-hardware

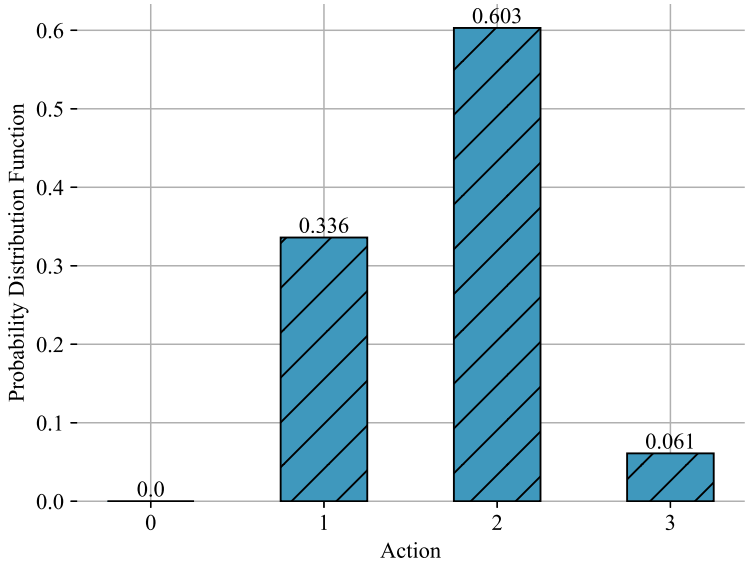


*Figure 6.5: Probability of occurrence of each action for **stable** context.*

decrease the $cwnd$ in steps or exponentially. But it does have a higher probability of choosing action 2, which helps to maintain stable throughput without extensively reducing the data transfer with the increase in the number of flows. Similar expected outcomes can be seen for the relative **decrease** context shown in Figure 6.6.

The trained algorithm was tested on a ZOTAC node connected to an AP (a DSS node, configured as AP, using Wi-Fi 802.11n) as end device in w-iLab.2 testbed. Using other end devices connected to the same AP, the number of flows in the Wi-Fi network was varied. The Figure 6.8 indicates the behavior of $cwnd$ for changing relative context values. When there is a relative increase in the number of flows in the network, the $cwnd$ is decreased based on the number of packets inflight. Similarly, when there is a relative decrease in the number of flows in the network, the $cwnd$ is increased based on the number of ACK_p.

The existing CC algorithms only notice the presence of other flow based on the packet loss [11], thus reducing the window size extensively to accommodate another flow. In contrast to this, the designed relative context-based RL algorithm still increases the $cwnd$ in smaller steps to maintain the throughput. Additionally, it was noticed that even with so many fluctuations in the number of flows, the packet loss was as low as 4 packets.

As mentioned earlier, one of the major drawbacks of this solution is that the

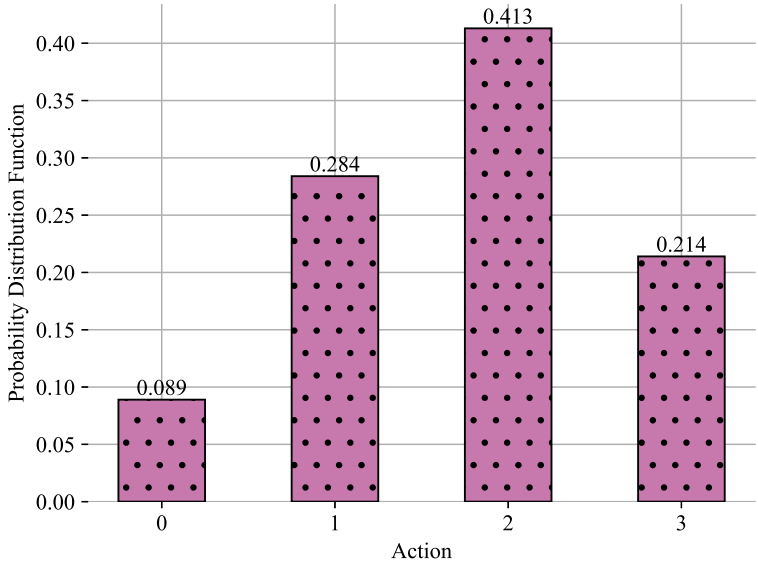


*Figure 6.6: Probability of occurrence of each action for relative **decrease** context.*

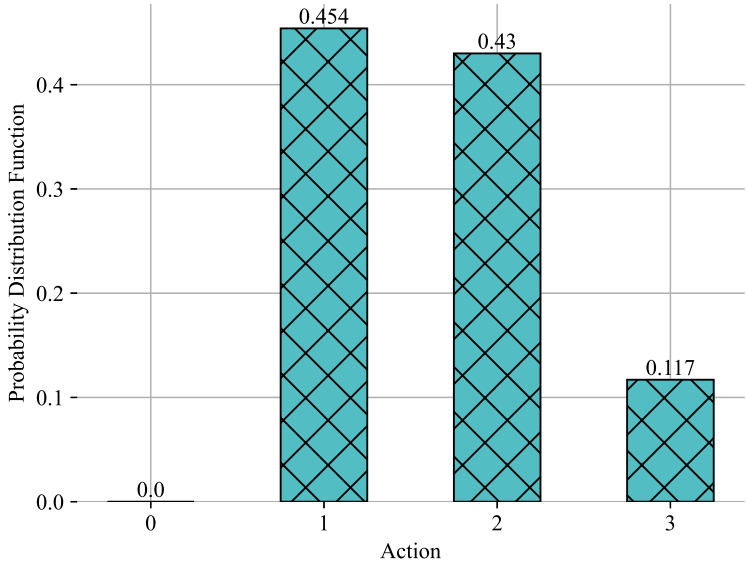


*Figure 6.7: Probability of occurrence of each action for relative **increase** context.*

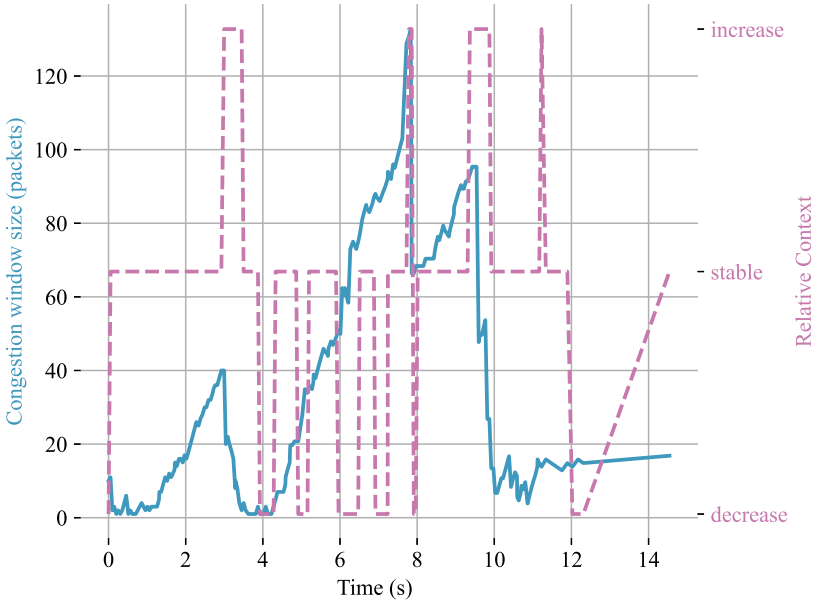


Figure 6.8: Behaviour of cwnd for relative increase or decrease in the number of flows in the network.

algorithm doesn't consider to what extent there has been a relative increase or decrease in context. One solution to this is changing the ϵ relative to the change in the context.

6.6 Conclusion

Private-professional wireless networks are widely incorporated in automation of industry, manufacturing, energy, etc. These networks are expected to cater to the strict throughput and latency demands of diverse applications. The future goals of wireless protocols involve integration of the application-network layer, continuous network monitoring, and adaptation of AI. With the influence of these technologies, this chapter proposes a multi-context-aware RL-based CC algorithm for private wireless networks. The INT-based CC algorithm is modeled as CMDP, and three different RL-based solutions are proposed. A fixed context-based and relative context-based solution using the Q-learning approach are implemented, trained, and tested on commercially available wireless network devices. The proposed designs and results assure the feasibility of integration of real-time network context and AI in the existing private wireless networks whilst catering to the needs of various applications.

The NACC designed in Chapter 3 focuses on providing differentiated services to applications along with CC. The RACC and FCCC algorithms proposed in Chapter 4 and Chapter 5 aim at achieving airtime fairness in Wi-Fi networks. Each of these algorithms uses different aggregated information from the intermediate network node in deciding the $cwnd$. Of these algorithms, the FCCC algorithm is proactive in nature, and the others are rule-based. To further anticipate the future network conditions, at the same time catering to the diverse needs of applications, the algorithms proposed in this chapter are considered. The algorithms introduced in this chapter consider the wireless network conditions as context. Though the results presented are for a limited number of contexts, the multi-context-aware solutions introduced can be further extended to incorporate application requirements and implemented in future private-professional networks.

References

[1] F. Paolucci, F. Cugini, P. Castoldi, and T. Osiński. *Enhancing 5G SDN/NFV edge with P4 data plane programmability*. IEEE Network, 35(3):154–160, 2021.

[2] 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 Transactions on Network and Service Management, 18(1):627–641, 2020.

[3] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke. *Tighter application-network interfacing to drive innovation in networked systems*. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, pages 53–57, 2021.

[4] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal. *Creating complex network services with ebpf: Experience and lessons learned*. In 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), pages 1–8. IEEE, 2018.

[5] A. Zappone, M. Di Renzo, and M. Debbah. *Wireless networks design in the era of deep learning: Model-based, AI-based, or both?* IEEE Transactions on Communications, 67(10):7331–7376, 2019.

[6] H. Jiang, Q. Li, Y. Jiang, G. Shen, R. Sinnott, C. Tian, and M. Xu. *When machine learning meets congestion control: A survey and comparison*. Computer Networks, 192:108033, 2021.

[7] K. Midhula and P. A. R. Kumar. *An adaptive congestion control protocol for wireless networks using deep reinforcement learning*. IEEE Transactions on Network and Service Management, 21(2):2027–2043, 2023.

[8] H. Jiang, Y. Luo, Q. Zhang, M. Yin, and C. Wu. *TCP-Gvegas with prediction and adaptation in multi-hop ad hoc networks*. Wireless Networks, 23:1535–1548, 2017.

[9] X. Liu, B. S. Amour, and A. Jaekel. *A reinforcement learning-based congestion control approach for V2V communication in VANET*. Applied Sciences, 13(6):3640, 2023.

[10] K.-L. A. Yau, P. Komisarczuk, and P. D. Teal. *Reinforcement learning for context awareness and intelligence in wireless networks: Review, new features and open issues*. Journal of Network and Computer Applications, 35(1):253–267, 2012.

[11] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke. *Adaptive transport layer protocols using in-band network telemetry and eBPF*. In 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pages 241–246. IEEE, 2021.

[12] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke. *Feedback-Based Control Loop Congestion Control Algorithm for Wireless Networks*. IEEE Access, 2024.

[13] C. J. Watkins and P. Dayan. *Q-learning*. Machine learning, 8:279–292, 1992.

[14] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. *Restructuring endpoint congestion control*. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pages 30–43, 2018.

7 Conclusion and Future Work

"A person is made by their belief. As they believe, so they are."

– Bhagavad Geeta

7.1 Conclusion

Connectivity has become crucial in today's world, and it is now embedded in key sectors such as industry, manufacturing, healthcare, finance, and the military through the adoption of private-professional wireless networks. This growth in connectivity, combined with applications like cloud computing, AR/VR, and human-machine interaction, has led to diverse demands on throughput, latency, and time-sensitivity, resulting in heterogeneous traffic flows in networks. The wired domain has adopted SDN, separating the control and data planes, enabling remote reconfiguration. However, fully leveraging this programmability requires real-time understanding of network states and application needs. Current networks offer limited quality-of-service options and static forwarding priorities, lacking sufficient granularity. The scheduling algorithms of the existing time-sensitive networking systems can only cater to the needs of the time-critical flows. As network traffic scales, maintaining differentiated treatment for each flow becomes challenging, going beyond the capacity of network components. At the same time, end devices are becoming more powerful and capable of analytics and AI-driven decision-making.

In this doctoral thesis, we take a different outlook at this problem from a higher network layer protocol standpoint. Transport layer protocols are designed to pro-

vide logical host-to-host communication, along with which some of them provide services like reliable, ordered, error-free data transfer using congestion and flow control mechanisms. But the existing transport layer protocols, or application protocols, consider only a limited perspective on the network and its characteristics. The programmability of such protocols and the connection of end-to-end behavior with the underlying network state have garnered attention recently, but there is still much to learn about the potential of implementing such techniques to help the network, especially wireless links, provide better treatment of application flows. They often rely on limited feedback and are trained offline, making them less adaptive to dynamic wireless environments. Leveraging INT, APP-NET interaction, and data programmability of the forwarding plane, the thesis proposes next-generation adaptive transport layer protocols based on network context and distributed knowledge.

Firstly, the relation between congestion window size and real-time INT parameters was studied. eBPF was integrated with INT to determine the right congestion window size based on the INT parameters. The data was collected for different wireless network conditions, also by inducing intentional packet loss in the wireless link. The results indicated a clear relation between the congestion window size and the parameters from INT, such as available queue space and packet loss events. Further, the CUBIC congestion control algorithm was modified to address packet loss due to wireless links. The throughput values indicated that the modified algorithm performed better than the CUBIC. The modified algorithm could maintain higher throughput values even with the higher link loss percentages. In the case of a wireless link with a 20% packet loss ratio, the new designs using real-time INT data achieved seven times higher throughput than the existing one. Additionally, when the progression of congestion window size in the absence of wireless loss was evaluated, all the modified algorithms reached the optimal window size quickly based on the intermediate node conditions and maintained the stable window size throughout the data transfer. The outcomes assure the flexibility of changing the application data transfer rate in real-time, allowing the design of adaptive transport protocols based on application requirements, resulting in tighter interaction between the transport layer and the network.

Utilizing the relation between INT and congestion window size, a rule-based network- and application-aware adaptive congestion control algorithm was designed. The algorithm was designed based on the real-time network context and distributed knowledge on aggregated flow information. In a multi-flow network design, the algorithm accomplishes service differentiation among various flows by taking into consideration the APP-REQ and its priorities. It offers data transmission service without congestion in a distributed wireless network design. The algorithm was initially tested for its responsiveness towards changing intermediate network conditions such as decrease in queue buffer space, increase in packet arrival rate,

and number of flows. Further, the algorithm was validated against the CUBIC congestion control algorithm. With the increase in the load sent, the throughput of the designed algorithm remained stable and was approximately three times better than that of CUBIC. The behavior of congestion window size also indicated that the designed algorithm reached stable size faster and additionally maintained the stable window size throughout. The algorithm was tested for its service differentiation for three flows with different priorities. The CUBIC congestion control algorithm is unaware of the requested application priorities and achieves the same throughput, whereas the designed algorithm provided throughput proportional to the priorities of the flows by modifying the application data transfer rate. The algorithm achieves guaranteed throughput by fixing the lower bound of congestion window size based on the priority.

Despite Wi-Fi standards promising high data throughput, the presence of an end device with very low data link quality can hinder the throughput of other end devices due to fair channel access of Wi-Fi. Therefore, there is a need for an algorithm to determine optimal data transfer rates without sacrificing the throughput of devices with higher physical data rates. This problem was resolved by designing a reactive and adaptive congestion control algorithm that adjusted the application data transfer rate depending on real-time network and airtime inputs. The intermediate node only communicated the airtime allocated for each end device. A benchmark comparison was made between the implemented design and the CUBIC congestion control algorithm. Using the reactive algorithm, the high physical data rate device's throughput improved by an average of 72% as compared to CUBIC. As little as 15%, on average, separated the airtime of the devices with the lowest physical data rate from those with the greatest, on an average of 30%. The reactive nature of the algorithm results in underutilization of the wireless channel in certain instance and can be improved.

Instead of acting reactively based on the observed network state, it is essential for the algorithm to act proactively. Therefore, a control loop-based congestion control algorithm was designed. To provide aggregated airtime feedback to each end device, the system further records the airtime use of each end device in AP. This additional accounting is then added to the INT header. To determine the appropriate congestion window size, the developed method considers the real-time aggregated airtime feedback from AP and network state information collected by INT. The proposed method was compared to the current CUBIC algorithm, and when the created algorithm was used instead of the CUBIC algorithm, end devices with greater physical data rates experienced a 58% increase in throughput. There was just an 18% variation in airtime between the end devices. The algorithm was validated for its sensitivity to the varying timeslot length and robustness to the changing physical data rate. The design was also tested for a potential use case of priority-based airtime fairness for future use cases.

Further, the learning-based methods were explored to design congestion control algorithms. Three different multi-context-aware reinforcement learning-based congestion control algorithms were proposed. The total throughput and latency for a data transfer of an end device are greatly impacted by wireless network characteristics such as channel quality, other competing nodes, and quantity of flows, which are independent of the actions taken by the end device. Therefore, the solutions were modeled as contextual Markov Decision Processes. Two of the proposed solutions were implemented using Q-learning. For the solution with fixed context values, the algorithm was tested on two devices with different physical data rates. Unlike the CUBIC algorithm, where the throughput is the same irrespective of the physical data rate, the contextual reinforcement learning algorithm achieves throughput proportional to the physical data rate. For the solution with relative context values, the algorithm was tested by varying the number of competing devices in the network. The designed algorithm adapts its data transfer to changing numbers of devices by modifying the congestion window size while maintaining stable data transfer.

All the algorithms presented in the thesis were implemented on commercially available wireless devices setup in a wireless environment in WiLab2 of IDLab. This promises the feasibility of adaptive algorithms in existing end devices. A brief summary of each of the designed algorithms is shown in Table 7.1.

The algorithms address the challenges of private-professional networks by moving the decision-making to the transport layer of end devices. This is achieved by adapting the application data transfer at the end device based on the real-time network context and aggregated flow information from intermediate network nodes. The end devices use rule-based, proactive, and learning-based approaches. The designed algorithms not only provide congestion-free data transfer but also have tighter interaction with applications and networks, thus providing different quality of service, airtime fairness, robustness to competing devices, and adaptability to the changing wireless network. Having fine-grained service differentiation and airtime fairness at the end device, the load on the intermediate network nodes is reduced. The designs assure the feasibility of adaptive applications and transport layers for future private-professional networks.

Table 7.1: Summary of the designed congestion control algorithms

Comparison feature	**NACC**	**RACC and FCCC**	**C-RL**
Network information from INT	Queue, packet arrival rate, aggregated flow information	Aggregated airtime information	Packet arrival rate, aggregated airtime and flow information
APP-REQ	Used to share application requests	-	Currently not used, to be used in the future
Aggregated information from the intermediate network node	Percentage of capacity share per flow basis, shared by intermediate network node	Percentage airtime share per end device basis, shared by AP	No such information
Network scenarios	Private-professional wireless networks	Wi-Fi networks	Private-professional wireless networks
Advantages	Application requirements are considered, no fixed service differentiation levels, available channel capacity shared based on priorities, bottleneck network conditions are considered, algorithm can be extended to any application	Airtime fairness, suitable for scheduled operation as it can introduce traffic shaping. Such scheduling could create ideal periods during which components might be turned off to save energy	Adaptable to wireless conditions such as changing data rates, fluctuating flows, etc. Single algorithm for all purposes, each application scenario can be added as plugins.
Disadvantages	Rule-based algorithm, the algorithm must be changed to add new features	The design is per end device, the airtime usage must be internally distributed at each end device for each flow.	Difficulty to emulate all the wireless network scenarios for training.

7.2 Future Work

Though a few of the challenges are addressed by the proposed designs of congestion control algorithms, there is a lot more to be explored.

7.2.1 Large-scale evaluation of the designed algorithms

The algorithms designed in this thesis are implemented on commercially available wireless hardware. It enables the possibility of implementation of the algorithms in a real-case scenario. But when the algorithms had to be tested on large networks, it puts a limit on the number of devices and traffic flows on which it can be tested. Since scalability is a very important feature of private-professional networks, the tests conducted in the thesis are not sufficient. In the scenarios of large-scale settings, tests can be conducted in a simulator, which is potential future work.

7.2.2 Dealing with multiple, potentially conflicting requirements

Currently the algorithms designed in the thesis focus on solving individual challenges by designing separate congestion control algorithms, for example, for bandwidth priority, airtime fairness, etc. But the thesis does not focus on integrating these features into a single algorithm. The thesis does explore the learning algorithms to address the same, but the results presented are preliminary, and the solution to support multiple flows per end device is still to be explored.

7.2.3 More research on learning-based algorithms

The final chapters of the thesis explore proactive and learning-based algorithms to anticipate the future network trend. The thesis proposes a learning model with three possible solutions for wireless private-professional networks. Only two of these solutions are preliminary implemented and tested. To extend the design to support varied applications, the context and state space get bigger and pose additional challenges on memory and computational management. The results prove to be dynamic enough for a very small network architecture. They do not yet prove sufficient for large-scale complex dynamic network environments, especially with the presence of time-critical flows. Potential future works include pattern recognition to reduce the state space, bringing more determinism in the behavior of the higher layer protocols, etc. Additionally, the open-source Openwifi platform can be leveraged to study the relation between transport layer parameters and medium access control parameters such as buffer space, contention window size, etc. Utilizing these can further reduce the state space, helping the algorithm to only use the key parameters of the overall network (from INT) and physical layer conditions of the device itself. This can also enable the possibility of designing

future transport layer protocols that enable certain features of the congestion control algorithm based on the application requirements using the physical layer and real-time network context. The transport layer protocol can act as a plugnized protocol that can adapt to providing different services, including time-sensitivity to time-critical flows or high bandwidth data rates for applications like audio/video.

7.2.4 Adaptive applications

To deal with the complex requirements of demanding applications, additional features must be taken into consideration, and even the application might need to be involved for adjusting its operation. An adaptive protocol can be designed to accommodate advanced features that can be activated upon the requirement and adjusted to the context. Also, adaptive capabilities of potential applications can be taken into consideration in the design.

7.2.5 Extension to QUIC

The congestion control algorithms designed in this thesis are for TCP; however, QUIC also uses congestion control algorithms, but it is not very mature. QUIC currently uses different internal priority-based queuing to avoid congestion and transfer only the essential data. The adaptive algorithms proposed in this thesis can do the same in a decentralized fashion on all the end devices. Hence, similar congestion control design concepts can be further extended to the QUIC transport layer protocol as well.

7.2.6 Additional capabilities to congestion control algorithms

Though the thesis had the intention of providing congestion-free service in wireless environments, the designed algorithms can adapt to the changes of wireless links, providing airtime fairness and differentiation of service. This promises further capabilities such as determinism, time sensitivity, etc. that can be integrated in congestion control algorithms.

7.2.7 Standardization and security aspects

Allowing programmability of intermediate network nodes does address several issues of the private-professional networks, but at the same time opens doors for vulnerabilities that can be abused by external actors. The learning-based algorithms used in the end devices can also abuse the network by intentionally using higher priorities for less critical applications. Hence there should be a specific standardization for each of these aspects defining the borders of programmable and learning-based extensions.

7.2.8 Challenges of INT in wired networks

Although INT is still achievable at high line rates in wired networks utilizing specialized network interface cards, its implementation at other very high rates is still challenging because of scalability, overhead, and hardware limitations. It is necessary to carefully weigh the trade-offs between network performance and telemetry granularity, as well as the capabilities of the underlying hardware and network architecture, in order to determine whether deploying INT in such settings is feasible.

A

Tighter application-network interfacing to drive innovation in networked systems

In this chapter, the architecture of an application-network interface that enables networks to share monitoring and feedback data with applications and allows applications to communicate traffic and monitoring needs to the network is described. This framework is used in implementing the CC algorithms discussed in this thesis.

⋆ ⋆ ⋆

This chapter is based on:
Tighter application-network interfacing to drive innovation in networked systems

Published in Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration (NAI '21), August 2021.

⋆ ⋆ ⋆

A.1 Introduction

Traditionally, applications are designed to be network agnostic, without knowing in real-time much about the network capabilities or the network's ability to satisfy their requirements. As such, the network behaves like a black box towards applications while being not aware of application requirements. At most, what networks can achieve is to distinguish between different traffic classes, a behavior that is not sufficient to treat increasingly diverse application requirements. Currently, network and application control planes are separated and are performed by different entities in the network. In some approaches, the network design is coupled to a specific application and some of the application logic is moved to the network nodes (like the *"in-network computing"* concept [1]). However, in many use cases, the application types and instances in the network can change over time and cannot be addressed by such an approach. Alternatively, the network can expose an API towards applications [2, 3] for expressing their requirements. Still, the network configuration is performed by a central entity that can hardly be dynamic enough to deal with changes in the application requirements. Also, applications can neither verify network performance nor can enforce certain network configurations directly.

In many dynamic use cases, like in AR/VR applications, wireless time-sensitive applications (AGV communication, assembling-line communication, robot communication), etc., integration between applications and networking is desirable. Moreover, integration between the data plane and the control plane is still a challenge. As such, application requirements should be shared via the data plane while network nodes can react (e.g. reconfigure scheduling) to such requirements and enrich the data plane with real-time monitoring capabilities. On the other hand, higher layer protocol logic can be optimized based on real-time network feedback. In order to move towards real-time network application integration, novel ways of interacting and exchanging a variety of information need to be defined, involving both the end devices and the network nodes.

For utilizing network reconfiguration and higher layer optimization for improved application performance real-time application-network integration needs to be in place, (i) where application data and control plane are integrated, by carrying and performing both application (APP) requirements and monitoring in-band in real-time, and (ii) where network nodes and application understand and take actions based on such information.

In this chapter, we present a network-application integration approach for private professional wireless networks, where both end devices and network nodes can be controlled. The following main contributions are made. (i) We design and implement the Application Network Agent (ANA) that enables applications to pass on real-time their traffic requirements and their monitoring needs, and enables the network to provide feedback towards applications. (ii) We design the logic

of network nodes to process application requirements as well as monitoring and feedback information. (iii) We evaluate the design in two different case studies.

A.2 ANA Design and Implementation

A modular design integrating the application's data plane and the control plane is achieved by placing an Application Network Agent (ANA) in the control path between the APP and the network stack (NET). Such design leads to a network and system independent APP-ANA interface and a network stack specific ANA-NET interface. Via the APP-ANA interface, applications can pass their information regarding their identification, traffic requirements and monitoring needs to the network, while the network can pass back information towards applications, such as the monitored performance of the traffic flow as well as monitoring feedback from the other end node.

Figure A.1 shows the architecture of ANA and its interaction with the applications and the network. ANA has a modular architecture that consists of a number of adapters that enable communication towards multi-plane network stack in southbound and towards in northbound.

Transport adapter offers interaction between ANA and the transport layer. For end-to-end communication, transport protocol parameters are crucial, especially in the case of connection-oriented transport protocols. As such, the transport adapter enables exposure of the TCP layer information (e.g. congestion and flow control window size) while in the north-south direction permits application to adapt TCP behavior.

Monitoring adapter interfaces ANA with the in-band network telemetry (INT) plane. The monitoring plane is implemented as an in-band feature in the data plane, where certain data packets will collect monitoring info in an end-to-end and in a per-hop and per-flow fashion, with support for wireless links as well [4]. As we consider only private professional networks that do not expand to a large number of network hops, introduced INT overhead is limited [5].

Feedback adapter interfaces ANA with the feedback plane to receive the monitored information from the in-band monitoring and feedback plane to be passed via the northbound interface towards application(s). Two different types of feedback are distinguished: feedback of data that was collected for the traffic flow with the node as destination and the feedback of data sent from the destination node to the traffic flow initial node. In the first case, the feedback is local as the monitored data are already available in the network monitoring stack. In the second case, the feedback data are received in-band via the reverse path from the destination node. Application requirements, monitoring as well as feedback data are encapsulated in the IPv6 extension header and reported in-band using data packets. To reduce the overhead, the application requirements and feedback data are first encoded using

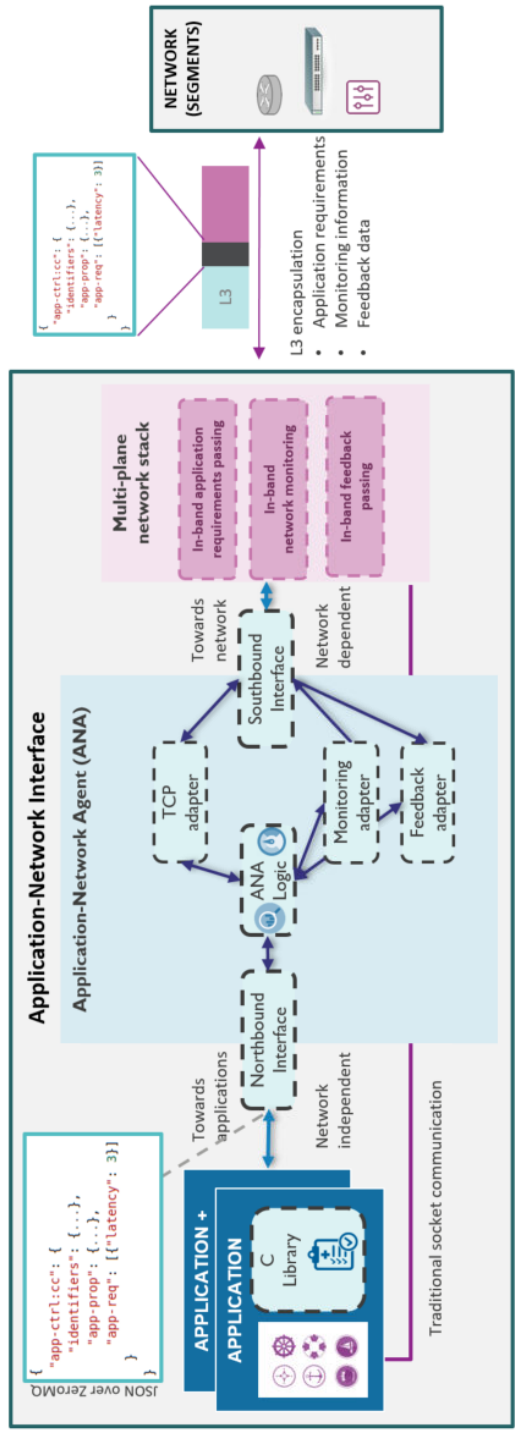


Figure A.1: Application Network Agent (ANA) architecture and its integration within network.

Concise Binary Object Representation (CBOR) before encapsulation.

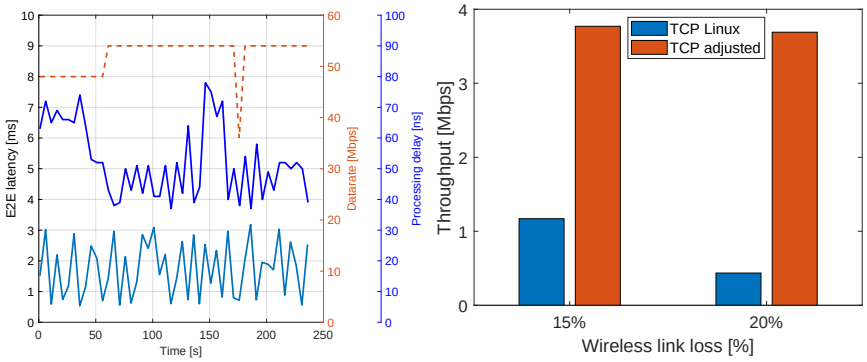
Exchanged data between ANA and applications and network, respectively, are modeled as JSON data structures. The data structure that ANA passes towards the monitoring stack and the network consists of three main parts: *the APP identifiers, the APP properties* and *the APP requirements*. The *APP identifiers* include the node ID identifying uniquely the traffic flow initial node, the APP ID identifying uniquely a type of application on that node, and the namespace ID that identifies uniquely each traffic flow of the same APP. The *APP properties* describe the properties of the traffic generated by the application (parameters such as traffic periodicity, payload size, etc.), while the *APP requirements* contain the list of required parameters to be fulfilled by the network. Similarly, the data structure that the network pass to ANA contains *the APP identifiers* to which the monitored data are linked, *the network path* the monitoring packet followed, and the *INT measurement* part. The *INT measurement* part will contain the end-to-end and hop-by-hop monitored data.

ANA logic processes the data received from both ends and creates the respective JSON data structures. In the southbound direction, the monitoring stack will encapsulate APP requirements JSON data structure into the data packet for injecting it into the network and to the destination node. Based on application requirements, ANA logic will instruct the monitoring stack on which flows and which monitoring parameters to monitor, e.g. for latency requirements ANA asks for monitoring of Rx/Tx time-stamping at each network hop, for reliability requirements it asks to monitor losses on a per-hop basis, etc. In the other direction, ANA will receive the INT monitoring feedback as a JSON data structure, extract the necessary information and pass it towards the application.

Application code changes and technical overhead should be limited to enable focusing on high-level features. Current applications interact with the network stack by opening a communication socket via which the datagrams are being passed. This traditional interface is extended with a unified, network-independent interface towards ANA, realized through ZeroMQ[1] (ZMQ) messaging. Currently, to integrate this interface with existing applications, a C library is provided that can be embedded with a few lines of code. The ZMQ messaging entity is used to interconnect the ANA logic with adapters and APPs. ZMQ has been chosen as a lightweight implementation of messaging library that supports different messaging patterns, as well as broker-based or broker-less PUB/SUB. The ZMQ communication is performed only locally between services inside the node, and for short message lengths (what is the case with ANA data structures) the added latency [2] is negligible compared to E2E communication latency.

[1]ZMQ documentation: https://zeromq.org/

[2]ZMQ effect of copying on latency: http://wiki.zeromq.org/results:copying



(a) Monitoring instruction *(b) Feedback-based TCP adjustment*

Figure A.2: Impact of ANI for different scenarios.

A.3 Evaluation

In this section, we demonstrate the usage of ANA in two given examples where performance and novel functionalities are achieved. Also, we compare the packet processing overhead with other SoTA solutions.

A.3.1 Case studies

A.3.1.1 Network monitoring and configuration

Network monitoring and configuration instruction can be controlled by the applications sharing their requirements in-band. An application can pass its requirements via the following command to ANA:

```
zmq::socket_t ANA_socket;
zmq::message_t msg;
...
msg.build(src_IP,src_port,dst_IP,dst_port,trans);
bool flag = ANA_socket.send(msg,latency);
```

In this case, the application opens a ZMQ socket towards ANA and passes its identifiers together with the requirements (in this case end-to-end latency). Once ANA receives the application requirements, ANA will instruct the monitoring plane about the traffic flow to be monitored, based on the *APP identifier* part, as well as parameters to be monitored, based on the *APP requirements*. As such, the monitoring plane will collect timestamping and processing latency on each network hop by setting the monitoring bitmap [5]. Using this telemetry data, the end-to-end latency and jitter can be determined. In addition to this, based on throughput

requirements, ANA can instruct the monitoring plane to collect info on the data rate of the wireless channel.

In an E2E wireless-wired (1 wireless link and 2 wired links) network topology in w-iLab.2 testbed, the network is instructed to monitor and to maintain a certain level of E2E latency (3 ms $<$) for certain time-sensitive traffic flow. Note that each network node supports time-aware shaping that is configured based on in-band application requirements. Figure A.2a shows the E2E latency of a time-sensitive network flow under network saturated condition with a coexisting UDP stream. Note that only the wireless hop monitoring information and E2E latency for time-sensitive flow are shown.

A.3.1.2 Feedback-based adjustment

Feedback-based adjustment of higher layer protocols based on the monitored data can be achieved via ANA. It is known that the CC mechanism of TCP can not distinguish between packet loss due to wireless issues or network congestion. Regardless of the packet losses cause, the TCP congestion mechanism will reduce the amount of data sent. Real-time detailed monitored data are used to distinguish between such events, so the application can instruct and maintain the TCP throughput under wireless losses.

Figure A.2b shows the throughput comparison between the default TCP CC mechanism in Linux with the adjusted control mechanism based on feedback data from the monitoring plane, for different wireless link loss ratios. The measurements were carried out in a multi-AP network topology setup with two wireless and one wired hops in between the end nodes. It is seen that for the cases with a higher packet loss ratio in wireless links, the feedback monitored data helps the TCP congestion mechanism to maintain the throughput. In the case of a wireless link with a 20% packet loss ratio, the adjusted approach based on the monitored data achieves six times higher throughput than the default Linux TCP congestion mechanism.

A.3.1.3 ANA Packet processing capabilities

can be compared with other SoTA solutions, to evaluate its performance. For performing such measurements we generate data packets that encapsulate application requirements and/or INT monitoring information in addition to the data payload. Measurements are performed in a setup where two nodes (3.3 GHz CPU, 256 GB RAM running Ubuntu 18) are connected by wire. The wired setup is used for its capabilities to support higher data rates, so we can stress test the proposed design. The first node will generate the INT enabled traffic while the other node will process such packets and reply with the monitored data. To compare our results with the results given in [6], we send 1000 Kpps, where packets are INT-enabled with 50 B

data payload. Measurements are done 50 times for 30 s long. Based on such measurements, ANA can process up to 998 Kpps compared to 855,573 Kpps in [6], resulting in only 2 ‰ throughput decrease compared to IPv6 traffic processing capabilities.

A.4 Related Work

Attempts to make networks application aware are not new [2, 7, 8]. In [2] the socket interface is extended with capabilities to support application demand sharing towards network stack. This way *socket intents* allow the application to share information about their communication patterns, however, the network configuration is still done via the control plane. Recently, application-aware IPv6 Networking [9] framework conveys application information into the network infrastructure. As such, the network can quickly adapt and perform the necessary network (re)configuration to maintain certain performance guarantees. Segmentation routing (SRv6) [10] is another approach to encode the forwarding rules inside the data packet, allowing the host application to choose the precise network nodes that will process the packet. In [6] a network API framework is proposed for application-aware traffic engineering (TE) using SRv6 that allows end-host applications to include a TE behavior inside the IP packet. Other solutions include ALTO [11], a protocol that provides applications with network state information, but does not include dynamic information and is not intended to substitute near-real-time protocols [11].

In addition to application requirements exposure functions, network monitoring functions need to be covered as well to support network-aware applications. The new approach of in-band network telemetry (INT) [12] that offers possibilities of fine-grained network monitoring is introduced recently. Such technique is extended to wireless networks as well [5] and offers full in-band network monitoring information exposure. MoWIE [13] includes integrated in-band and out-of-band network monitoring exposure to reduce the necessity of datapath packet format updates or end-host modification when full in-band is used.

In case of ANA integration of in-band monitoring and feedback is done transparent to the application side. Compared to [9] ANA encapsulates not only APP requirements but also real-time in-band monitoring and feedback data, enabling real-time network node configuration. ANA achieves the same real-time monitoring and network instruction as [6], but it decreases the throughput of IPv6 traffic only by 2 ‰, compared to $\sim$ 10% decrease in [6].

A.5 Conclusion

In this chapter, we presented a design approach for a rich and extensible application network interface. Such design, on the one hand, enables applications to pass their identification, traffic properties, and requirements, as well as monitoring needs towards the network, while on the other hand, the network can monitor the achieved performance of individual traffic flows and feedback such information towards applications. The ANA design offers a unified, network-independent interface for applications to express their requirements and to get feedback from the network.

The potential benefits of the proposed ANA architecture has been evaluated in two different use cases. In the first use case, ANA is used to enable network monitoring and reconfiguration according to the application requirements. The second use case demonstrates how feedback information from the monitoring plane can be used to adjust the transport layer behavior for achieving better throughput under wireless link losses. Next to this, the processing capabilities of the data plane are not affected in comparison to cases where such monitoring and feedback plane is absent and are 10% higher compared to other SoTA solutions.

References

[1] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman. *The case for in-network computing on demand*. In Proceedings of the Fourteenth EuroSys Conference 2019, pages 1–16, 2019.

[2] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann. *Socket intents: Leveraging application awareness for multi-access connectivity*. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, pages 295–300, 2013.

[3] G. V. Schueren, Q. De Coninck, and O. Bonaventure. *TCPSnitch: Dissecting the Usage of the Socket API*. arXiv preprint arXiv:1711.00674, 2017.

[4] J. Haxhibeqiri, I. Moerman, and J. Hoebeke. *Low overhead, fine-grained end-to-end monitoring of wireless networks using in-band telemetry*. In CNSM2019, the 15th International Conference on Network and Service Management, pages 1–5, 2019.

[5] 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 Transactions on Network and Service Management, 2021.

[6] T. Miyasaka, Y. Hei, and T. Kitahara. *Networkapi: An in-band signalling application-aware traffic engineering using srv6 and ip anycast*. IEICE TRANSACTIONS on Information and Systems, 104(5):617–627, 2021.

[7] B. Hesmans and O. Bonaventure. *An enhanced socket API for Multipath TCP*. In Proceedings of the 2016 applied networking research workshop, pages 1–6, 2016.

[8] K. R. Evensen, K.-J. Grinnemo, A. F. Hansen, N. Khademi, S. Mangiante, P. McManus, G. Papastergiou, D. Ros, M. Tüxen, E. Vyncke, et al. *The NEAT Architecture*. Online) https://www. neat-project. org, 2015.

[9] Z. Li, S. Peng, D. Voyer, C. Xie, P. Liu, C. Liu, K. Ebisawa, S. Previdi, and J. Guichard. *Application-aware IPv6 Networking (APN6) Framework*. Internet-Draft draft-li-apn6-framework-00, Internet Engineering Task Force, November 2019. Work in Progress. Available from: https://datatracker.ietf.org/doc/html/draft-li-apn6-framework-00.

[10] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li. *SRv6 Network Programming*. Internet-Draft draft-filsfils-spring-srv6-network-programming-07, Internet Engineering Task Force, February 2019. Work in Progress. Available from: https://datatracker.ietf.org/doc/html/draft-filsfils-spring-srv6-network-programming-07.

[11] S. Kiesel, W. Roome, R. Woundy, S. Previdi, S. Shalunov, R. Alimi, R. Penno, and Y. R. Yang. *Application-Layer Traffic Optimization (ALTO) Protocol*. RFC 7285, September 2014. Available from: https://rfc-editor.org/rfc/rfc7285.txt, doi:10.17487/RFC7285.

[12] F. Brockners, S. Bhandari, and T. Mizrahi. *Data Fields for In-situ OAM*. Internet-Draft draft-ietf-ippm-ioam-data-12, Internet Engineering Task Force, February 2021. Work in Progress. Available from: https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-data-12.

[13] Y. Zhang, G. Li, C. Xiong, Y. Lei, W. Huang, Y. Han, A. Walid, Y. R. Yang, and Z.-L. Zhang. *MoWIE: Toward Systematic, Adaptive Network Information Exposure as an Enabling Technique for Cloud-Based Applications over 5G and Beyond*. In Proceedings of the Workshop on Network Application Integration/CoDesign, pages 20–27, 2020.