High Throughputs and Information Freshness over the Internet via Transport Layer Advancements

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

BY

TANYA SHREEDHAR

PhD14103

ELECTRONICS AND COMMUNICATION ENGINEERING

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

NEW DELHI- 110020

JUNE 2022

High Throughputs and Information Freshness over the Internet via Transport Layer Advancements

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

BY

TANYA SHREEDHAR

PhD14103

ELECTRONICS AND COMMUNICATION ENGINEERING

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

NEW DELHI- 110020

JUNE 2022

THESIS CERTIFICATE

This is to certify that the thesis titled **High Throughputs and Information Freshness over the Internet via Transport Layer Advancements**, submitted by **Tanya Shreedhar**, to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of **Doctor of Philosophy**, is an original research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Sanjit Krishnan Kaul
Thesis Supervisor
Associate Professor
IIIT Delhi, 110020

Place: New Delhi

Dedicated to the loving memory of my father.

Not a single day passes by when I don't miss you.

ACKNOWLEDGEMENTS

"It takes a village to get a PhD", and I have been blessed to have a strong support system around me. I have met some amazing people during this journey who have significantly impacted my academic life. The words might fall short while I convey my heartfelt thanks to everyone who has helped me reach here.

First and foremost, I would like to thank my advisor, Dr. Sanjit Krishnan Kaul, for allowing me to work and learn with him. This thesis would not have been possible without his continuous support, guidance, and patience. He has given me immense freedom to pursue research areas that interest me. Sanjit's enthusiasm for research is magnetic, and he is one of the few advisors who always have their doors open for discussions. He has taught me the importance of returning to the board when stuck, and discussions with him have been some of my most cherished memories. He has profoundly affected my research perspective and has inspired, motivated, supported, and encouraged me to become a better researcher.

I am also extremely grateful to the great collaborators in my PhD. I want to thank Prof. Roy D. Yates (Rutgers University) for his unwavering support and encouragement throughout all these years. His great insights and valuable feedback have helped me better shape and structure my work. I also thank Prof. Jussi Kangasharju (University of Helsinki) for hosting me and giving me insightful feedback on the writing and motivation of my research articles. Jussi is a fantastic mentor and a lovely human being from whom I also learned many valuable life skills. I will also take this opportunity to thank Prof. Dr.-Ing. Jörg Ott (Technical University of Munich). He invited and hosted me at the Chair of Connected Mobility at Technical University of Munich (TUM), Germany. He was always kind and supportive, making my stay at the Chair comfortable even during the challenging conditions that the pandemic presented. He is one of the most hard-working Professors I have come across, and I admire his excellent eye for detail. I would also like to thank Dr. Vaibhav Bajpai, who pushed me beyond my comfort zone and made me appreciate measurement-based research. He is a kind and

compassionate person who helped me settle in the new environment in Germany. I will also like to thank my good friend and collaborator, Dr. Nitinder Mohan, who encouraged me as I entered the world of systems research. He has always inspired me to do good research. I also had the opportunity to collaborate with some great researchers - Aleksandr Zavodovski (Uppsala University), Otto Waltari (University of Helsinki), Oliver Gasser (Max Planck Institute for Informatics), Trinh Viet Doan (TUM), Mike Kosek (TUM), Ricky K. P. Mok (CAIDA/UCSD). I was fortunate to mentor many talented graduate and undergraduate students at TUM - Aakash Kamble, Rohit Panda, Sergey Podanev, Luca Schumann and Leo Eichhorn. I wholeheartedly thank them all for being a part of my journey.

During my PhD, I have been fortunate to meet and become friends with some fantastic people. Acknowledgments are due to my colleagues and friends at Wireless Systems Lab and Mobile and Ubiquitous Computing research group - Anupriya Tuli, Nipun Batra, Milan Jain, Alvika Gautam, Dheryta Jaisinghani, Sonia Soubam, Deepika Yadav, Garvita Bajaj, Haroon Rashid, Anil Sharma, Shivangi Agarwal and Gunjan Singh. They have been my pillars of support through various ups and downs. I will also like to thank my colleagues at the Chair of Connected Mobility - Vittorio Cozzolino, Pegah Torkamandi, Trinh Viet Doan, Mike Kosek, Linus Dietz, Leonardo Tonetto, Michael Haus, Aygün Baltaci, Ljubica Pajević Kärkkäinen and Wolfgang Wörndl. They welcomed me and made me feel at home.

I also want to express my gratitude to the administrative staff at IIIT-Delhi- Ms. Priti Patel, Ms. Anshu Dureja, Ms. Sheetu Ahuja and Mr. Ashutosh Brahma. They have always been kind and supportive of the students. I am also grateful for the immense support received from the staff at TUM - Ms. Christine Lissner (admin) and Mr. Simon Zelenski (tech). I appreciate and acknowledge the financial support I have received for my PhD research from various intuitions - TCS Research, Microsoft Research, Overseas Research Fellowship, ACM Student Travel Grant, LRN Foundation International Travel Grant Award and NSF Student Conference Award. This support has been crucial for the research presented in this thesis.

Finally, this PhD would have been impossible without the support of my loving

family. I owe this thesis to my father, Mr. Deepak Shreedhar, who was my biggest champion. He has celebrated my smallest of successes and inspired me during my setbacks. Unfortunately, fate took him away from us, and I dedicate this thesis to him. My mother, Mrs. Suman Shreedhar, has always been my backbone. As an educator herself, she showed me that education is of utmost importance. Her hard work, sacrifices, perseverance, and resilience have always inspired me. I thank my parents for all that I am and all that I will be. I also want to thank my brother Surya and my sister-in-law Sumedha for their unwavering support, motivation, and encouragement at every step of my PhD journey.

Tanya Shreedhar

ABSTRACT

Next-generation applications such as augmented reality, virtual reality, teleoperated driving, and video streaming, along with a wide spectrum of real-time monitoring and actuation applications, are expected to challenge the current Internet on at least two fronts. Many of these applications desire high throughput and reliability at low end-to-end path latencies. To better support them, we must optimize the joint use of the diversity of high link rate wireless access technologies commonly available at end-user devices. In addition, there is a burgeoning class of applications that requires the availability of fresh information (for example, sensor measurements and actuation commands in IoT applications) at the destination. The current Internet treats such applications no differently than it does those that care for throughput. In this thesis, we address the challenges posed by these distinct requirements of high throughput and high freshness via innovations at the transport layer of the networking stack.

We address the challenge posed by applications that require high end-to-end throughputs via a novel cross-layer scheduler, QAware, for Multipath TCP (MPTCP). The QAware scheduler uses local queue occupancy information for every access network available on a user device in addition to the typically used end-to-end round-trip delay estimates. This results in a more efficient use of the available interfaces and considerable gains in aggregate throughput compared to other MPTCP schedulers for a varied set of applications and over heterogeneous access networks.

For real-time monitoring and remote sensing applications, we address the challenge of enabling freshness, as quantified by the metric of *age-of-information* (AoI) over an end-to-end Internet path. Specifically, we propose and detail the Age Control Protocol (ACP) and its improved version called ACP+. Both use `ACK`s to maintain an estimate of the number of unacknowledged packets in the system

along with the end-to-end RTT. This, together with an estimate of the time-average age of updates is used to determine an ACP source's update rate. We study the efficacy of ACP and ACP+ using extensive simulations and real-world experiments over the Internet. To gain further insight into age control, we also empirically compare ACP+ with a mix of loss-based, delay-based, and hybrid congestion control algorithms used by TCP. TCP tries to fill the network pipe using estimates of bottleneck rate and baseline RTT, but these estimates may not shed as much light on the age-optimizing update rate.

In our experiments over paths in the Internet, ACP+ utilizes only a fraction of the bottleneck link rate for achieving low age. When the path had a wireless access as its first hop that was the bottleneck link, the path beyond the access, with links much faster than the access, was the constraining factor with regards to minimizing age over the end-to-end path. Age being optimized at update rates much lower than typical bottleneck access link rates has interesting consequences for end-to-end flows sharing the wireless access to send updates to the cloud.

We experimented with a large number of ACP+ sources (up to 80 sources and fixed physical layer rates of 6, 12 and 24 Mbps) sharing a WiFi access point to send their updates to a cloud server over the Internet. We show that ACP+ allows sources to share the access well. When the wireless access isn't the constraining factor, given the low age minimizing rate over the end-to-end path, all sources send at the minimizing rate. As the number of sources increases, the resulting congestion over the WiFi access has ACP+ gradually reduce the rate of updates per source in a manner such that the sources together fully utilize the WiFi access link rate. While fully utilizing the access like TCP, ACP+, however, keeps age much lower than TCP congestion control algorithms. Even the packet retry rates because of collisions over WiFi are much lower than for TCP. In fact, TCP algorithms are unsuitable for age control, which we demonstrate using simulations and real-experiments in this thesis. On the other hand, ACP+'s behavior, as determined using controlled simulations, is in line with what would be expected of a good age control strategy enabling sharing of access amongst multiple sources.

User devices are expected to support a mix of applications, some of which may care for high throughput and others for the freshness of information. We

conclude this thesis with a study on the coexistence of ACP+ and TCP flows sharing an end-to-end Internet path over a WiFi access. In line with expectation, ACP+ flows coexisting with TCP flows remain unaffected when assigned a higher Differentiated Services Code Point (DSCP) priority when all flows originate in the same device. However, the gains from prioritization vanish when the flows are instead sharing a contended wireless access.

Contents

List of Tables

List of Figures

ABBREVIATIONS

ACK	Acknowledgement
ACP	Age Control Protocol
AIMD	Additive Increase Multiplicative Decrease
AoI	Age of Information
AP	Access Point
AR	Augmented Reality
BDP	Bandwidth Delay Product
CDF	Cumulative Distribution Function
CPS	Cyber-Physical Systems
CSMA/CA	Carrier-Sense Multiple Access with Collision Avoidance
CWND	Congestion Window
DSCP	Differentiated Services Code Point
EDCA	Enhanced Distributed Channel Access
HCCA	Hybrid Controlled Channel Access
IFS	Inter-Frame Spacing
IoT	Internet-of-Things
IP	Internet Protocol
MAC	Medium Access Control
MPTCP	Multipath TCP
OpenWRT	Open Wireless Router
PHY	Physical Layer
QoS	Quality of Service

QoE	Quality of Experience
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SMSS	Sender Maximum Segment Size
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
VR	Virtual Reality

PUBLICATION LIST

[1] **ACP+: An Age Control Protocol for the Internet**. Tanya Shreedhar, Sanjit K. Kaul and Roy D. Yates. IEEE/ACM Transactions on Networking 2022. DOI: arxiv.org/abs/2210.12539. [Under Submission]

[2] **A Longitudinal View at the Adoption of Multipath TCP**. Tanya Shreedhar, Danesh Zeynali, Oliver Gasser, Nitinder Mohan and Jörg Ott. IEEE/ACM Transactions on Networking 2022. DOI: arxiv.org/abs/2205.12138. [Under Submission]

[3] **Coexistence of Age Sensitive Traffic and High Throughput Flows: Does Prioritization Help?**. Tanya Shreedhar, Sanjit K. Kaul and Roy D. Yates. IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2022. DOI: 10.1109/INFOCOMWKSHPS54753.2022.9798247.

[4] **Evaluating QUIC Performance over Web, Cloud Storage and Video Workloads**. Tanya Shreedhar, Rohit Panda, Sergey Podanev and Vaibhav Bajpai. IEEE Transactions on Network and Service Management (TNSM), 2022. DOI: 10.1109/TNSM.2021.3134562.

[5] **An Empirical Study of Ageing in the Cloud**. Tanya Shreedhar, Sanjit K. Kaul and Roy D. Yates. IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2021. DOI: 10.1109/INFOCOMWKSHPS51825.2021.9484567.

[6] **From Single Lane to Highways: Analyzing the Adoption of Multipath TCP in the Internet**. Florian Aschenbrenner, Tanya Shreedhar, Oliver Gasser, Nitinder Mohan and Jörg Ott. IFIP Networking Conference (IFIP Networking), 2021. DOI: 10.23919/IFIPNetworking52078.2021.9472785.

[7] **Beyond QUIC v1 – A First Look at Recent Transport Layer IETF Standardization Efforts**. Mike Kosek, Tanya Shreedhar and Vaibhav Bajpai. IEEE Communications Magazine, April 2021. DOI: 10.1109/MCOM.001.2000877.

[8] **Distributed Ledgers for Distributed Edge: Are we there yet?**. Leo Eichhorn, Tanya Shreedhar, Aleksandr Zavodovski and Nitinder Mohan. Proceedings of the ACM Interdisciplinary Workshop on (de) Centralization in the Internet (IWCI'21), 2021. DOI: 10.1145/3488663.3493687.

[9] **An Age Control Transport Protocol for Delivering Fresh Updates in the Internet-of-Things**. Tanya Shreedhar, Sanjit K. Kaul and Roy D. Yates. 20th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE, 2019. DOI: 10.1109/WoWMoM.2019.8793011.

[10] **QAware: A Cross-Layer Approach to MPTCP Scheduling**. Tanya Shreedhar, Nitinder Mohan, Sanjit K. Kaul and Jussi Kangasharju. IFIP Networking, May 14-16, 2018, Zurich, Switzerland (2018). DOI: 10.23919/IFIPNetworking.2018.8696843.

[11] [Poster] **ACP: Age Control Protocol for Minimizing Age of Information over the Internet**. Tanya Shreedhar, Sanjit K. Kaul and Roy D. Yates. 24th International Conference on Mobile Computing and Networking (MobiCom), ACM, 2018. DOI: 10.1145/3241539.3267740.

[12] [Poster] **Redesigning MPTCP for Edge Clouds**. Nitinder Mohan, Tanya Shreedhar, Aleksandr Zavodovski, Otto Waltari, Jussi Kangasharju and Sanjit K. Kaul. 24th International Conference on Mobile Computing and Networking (MobiCom), ACM, 2018. DOI: 10.1145/3241539.3267738.

[13] [Manuscript] **Is two greater than one?: Analyzing Multipath TCP over Dual-LTE in the Wild**. Nitinder Mohan, Tanya Shreedhar, Aleksandr Zavodovski, Jussi Kangasharju and Sanjit K. Kaul. DOI: arxiv.org/abs/1909.02601.

[14] [Code] **Age Control Protocol - GitHub**. Tanya Shreedhar, Sanjit K. Kaul and Roy D. Yates. Code: github.com/tanyashreedhar/AgeControlProtocolPlus.

[15] [Code] **MPTCP QAware - GitHub**. Tanya Shreedhar, Nitinder Mohan, Sanjit K. Kaul and Jussi Kangasharju. Code: github.com/tanyashreedhar/mptcp-QueueAware.

[16] [Code and Dataset] **QUIC Performance Evaluation - GitHub**. Tanya Shreedhar, Rohit Panda, Sergey Podanev and Vaibhav Bajpai. Code: github.com/tanyashreedhar/tnsm-2021-quic.

[17] [Dataset and Scanning Service] **Multipath TCP Measurement Service**. Florian Aschenbrenner, Tanya Shreedhar, Oliver Gasser, Nitinder Mohan, and Jörg Ott. DOI: 10.14459/2021mp1610028. Scanning Service: mptcp.io.

Chapter 1

Introduction

The Internet has changed tremendously since its inception in the 1980s. Communication technologies that provide users access to the Internet have evolved rapidly over the last couple of decades. Not only do we have a wide variety of access technologies, for example, cellular and WiFi, but access technologies have also shown rapid improvements in link rates they support. Wireless access technologies promise data rates of Gbps and low latencies of the order of milliseconds. The Internet backbone can support hundreds of Gbps of traffic. In fact, researchers in Japan recently broke the data transfer record by achieving petabyte/s speeds over fiber optic network [3]. In addition, a diversity of access technologies are available within end-user devices. The user devices come equipped with multiple network interfaces that, in principle, allow them to utilize heterogeneous access networks simultaneously [4].

The support for faster end-to-end paths over the Internet has coincided with an increase in demand for next-generation applications such as augmented reality

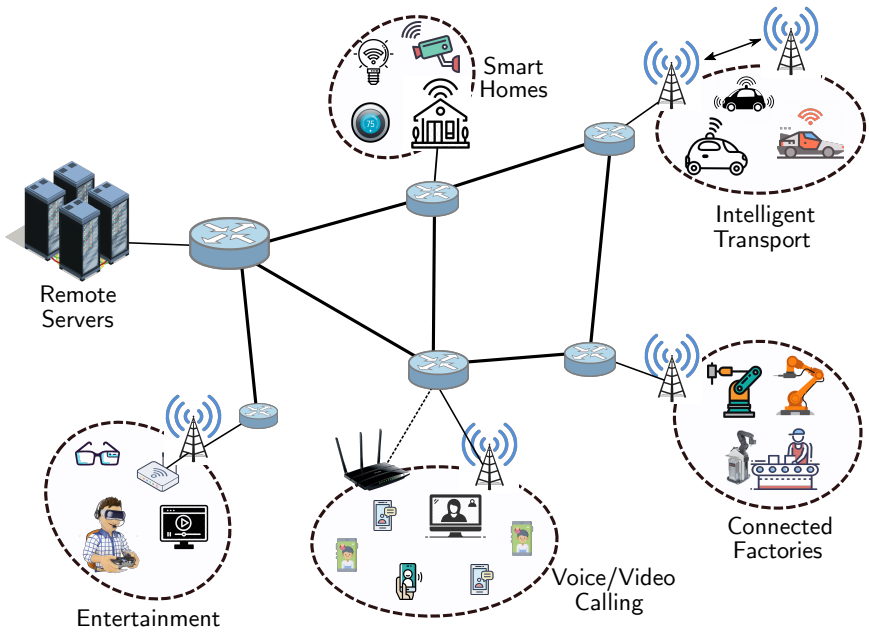


Figure 1.1: Applications with different QoS requirements connected to the Internet over different access networks.

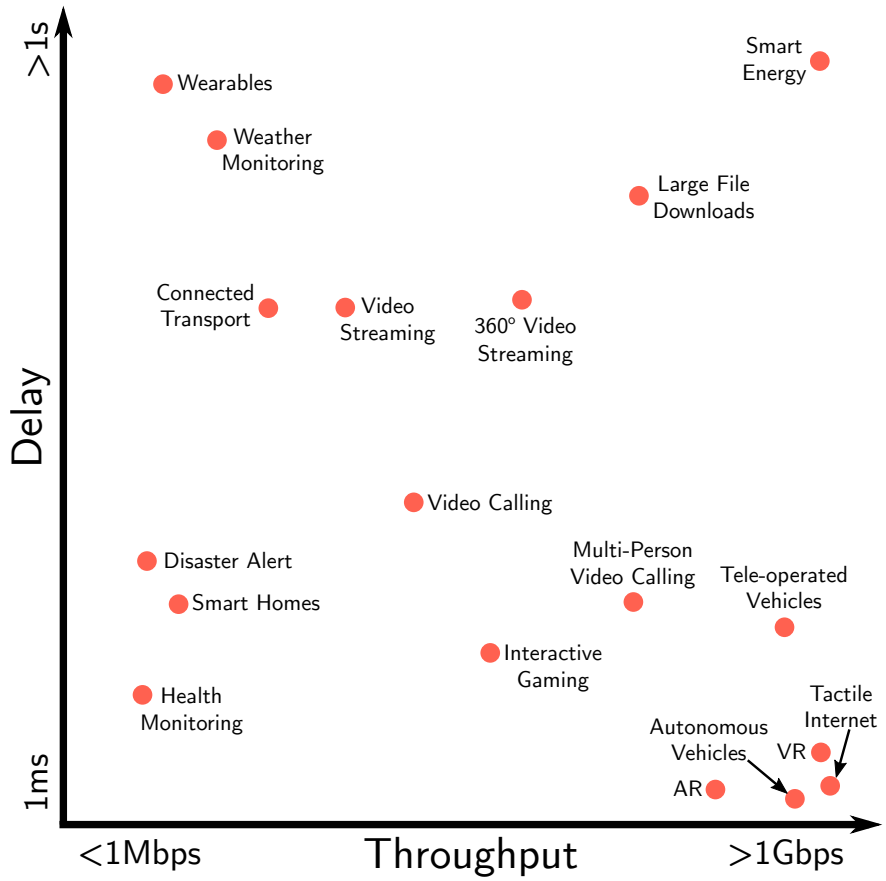


Figure 1.2: Internet applications network throughput and delay requirements for optimal operation.

(AR), virtual reality (VR), teleoperated driving, video streaming along with a wide spectrum of real-time monitoring and actuation applications that are often categorized under the Internet-of-Things (IoT) or cyber-physical systems. Figure 1.1 illustrates the diverse applications that typically share the Internet. These user applications have come far from traditional file transfer and voice/video calling. The global market share of these next-generation applications is expected to increase manifold by 2027 [5].

Not only has the sheer variety of applications burgeoned, but they also come with a diverse set of Quality-of-Service (QoS) requirements. Figure 1.2 illustrates differences in the throughput-delay requirements of a variety of current applications [6, 7, 8, 9, 10]. Content delivery applications, including video streaming, social media, and large file downloads, require high throughputs between the content-hosting server and the client for their satisfactory operation [11]. Also, while these applications desire reliable data transfer, they are often delay-tolerant [12]. Mean-

while, applications such as AR, VR, interactive gaming, and video calling require low end-to-end delays and high throughputs between the user and the compute server to maintain human immersiveness [13].

On the other hand, real-time monitoring and actuation, which spans a wide variety of applications including smart energy, health monitoring, smart homes, and weather monitoring, desires sensed information to be refreshed ever so often such that it is as fresh as possible at the monitoring facility in the cloud. The same applies to actuation commands sent on the reverse path from the monitoring facility to controlled devices. Such monitoring is often highly packet error resilient and may not impose very high throughput requirements (for example, if the sensed updates and actuation commands lead to small data payloads). As shown in Figure 1.1, all these next-generation applications desiring widely different QoS operate over shared network paths where coexisting traffic from multiple applications can result in significant performance degradation [14].

Transport protocols are responsible for supporting end-to-end communication between interacting end hosts. They help applications achieve their operational requirements while masking the underlying dynamic network conditions from them, essentially acting as a glue between the applications and the diverse networks they may use for communication. These protocols can provide features like reliability, flow control, congestion control that enables sharing of networks among many geographically distributed applications, and in-order delivery over an end-to-end path [15]. For instance, congestion control prevents user applications from flooding the underlying network, which could lead to significantly large packet drops and delays. Effective transport can improve the performance of different applications.

The majority of applications that use the Internet have relied primarily on two transport protocols – Transmission Control Protocol (TCP) [15] and User Datagram Protocol (UDP) [16], both of which were designed almost four decades ago. In fact, today, most applications use one of the two protocols, even though they might not be ideal to meet the QoS requirements of the applications [17]. As such, we identify two challenges in effectively supporting the operation of such applications on current networks. First, given that many such applications desire high throughput and reliability at low end-to-end path latencies, we must optimize

the use of the diversity in wireless access technologies available at end-user devices. Second, to support the burgeoning class of applications that rely on the availability of timely information, innovation in transport protocol is necessary to support end-to-end information freshness inherently.

In this thesis, we aim to bridge the gap between the existing transport layer solutions and the requirements of emerging Internet applications, given the availability of diverse communication technologies.

1.1 Research Contributions

We address the challenge posed by applications that require high end-to-end throughputs via a novel cross-layer scheduler, QAware, for Multipath TCP (MPTCP). Unlike the state-of-the-art MPTCP schedulers, QAware achieves significantly higher throughputs for a varied set of content delivery applications and over heterogeneous access networks.

Additionally, for information freshness desiring applications, in this thesis, we also address the challenge of enabling freshness, as quantified by the metric of age-of-information, over an end-to-end Internet path. Specifically, we propose the Age Control Protocol (ACP) and its improved version ACP+. We detail its performance over real-world Internet paths and shared wireless access. We compare and contrast it with many scheduling algorithms used over end-to-end paths by Transmission Control Protocol (TCP). Last but not the least, we consider the coexistence of ACP+ and TCP, as user devices will often support a mix of applications, some of which care for high throughput and others for the freshness of information.

1.1.1 QAware: A Cross-Layer Approach to MPTCP Scheduling

As discussed earlier in this chapter, content delivery applications desire an end-to-end connection with high throughput and high reliability between a client and a server. For example, the recommended average throughput required to support a 4K video stream is ≈14Mbps [18] and can become as high as ≈ 40Mbps for

streaming a 360° video [7]. Content delivery applications are also relatively delay-tolerant. TCP is the best fit for these applications as it efficiently fills the network pipes and ensures delivery of all the data since these applications are loss-sensitive.

As the end hosts like smartphones, laptops and servers are often equipped with multiple access interfaces such as Ethernet, WiFi and 3G/4G, it is possible to aggregate the bandwidth of multiple parallel network paths between communicating devices. Such a configuration is quite suitable for Multipath TCP (MPTCP) [19]. MPTCP, a standardized extension to TCP, allows end hosts to utilize multiple parallel paths for simultaneous data transfer. It achieves robustness and resilience to link failures and provides seamless connection handovers over different network interfaces [20]. MPTCP adds a scheduling layer over existing TCP connections between end hosts and routes application packets to one of the subflows based on a decision parameter. Existing schedulers typically use estimates of end-to-end path properties, such as round-trip-delay and bottleneck bandwidth, for making the scheduling decisions [21, 22]. In this thesis, we show that scheduling decisions can be significantly improved by also using readily available local information about the occupancy of device driver queues.

We propose *QAware* [23], a novel *cross-layer* approach for scheduling packets across all available MPTCP subflows. QAware's design is motivated by our experimental findings that combining local device driver queue occupancy with the traditional end-to-end delay measurements yields better throughput performance. We found that as a particular flow is used more, its end-to-end delay increases gradually, making it less attractive to use. However, the traditional, purely end-to-end path delay estimation-based approaches react very slowly to these changes. QAware leverages queue theoretical insights to create a scheduling policy that combines end-to-end delay estimates with local queue occupancy information. This results in more efficient use of the available interfaces and considerable gains in aggregate throughput. We design and evaluate QAware's performance through simulations and also through real experiments, comparing it to existing state-of-the-art schedulers. We make QAware open-source and publicly available at [24].

1.1.2 Enabling Delivery of Fresh Information Over the Internet

While innovations over MPTCP allow applications to maximize throughput across available paths, it does not help the performance of real-time monitoring applications that desire information freshness much more than throughput or reliability. In fact, none of the existing transport protocols like TCP, UDP, RTP and QUIC [25, 26], which support various applications like file transfer, video streaming, and voice applications, can support the requirements of such applications that desire *freshness* over an end-to-end Internet path. We measure freshness at a monitor using the metric of *age*. Age at the monitor is the time elapsed since the generation time of the most recently generated update that the monitor has received. When the monitor receives a more recently generated update, the age at the monitor is reset to the time elapsed between the generation of the update and its reception. In the absence of updates, the age at the monitor increases linearly with time.

To this end, we propose the *Age Control Protocol* (ACP) [27, 28, 29, 30], a novel end-to-end transport protocol that sits on top of UDP in the networking stack and enables end-to-end freshness in a network transparent manner. ACP adapts the rate of updates from a source to changes in the network, with the goal of minimizing the age of information, which is the time average of age, at a monitor. ACP does not make any assumptions about the network and does not expect any information from any other layer in the networking stack. ACP at the source node uses `ACK` (acknowledgment) packets sent to it from the ACP at the monitor to maintain an estimate of average delays over the end-to-end path using the round-trip time (RTT). In addition, ACP at the source maintains an estimate of the time-average count of update packets that have been sent by it but not yet acknowledged by the monitor. The above two estimates, together with an estimate of the time-average age of updates at the monitor, are used by the source ACP to update the rate of sending updates over the end-to-end path.

In the thesis, we detail ACP's age control algorithm. We study its efficacy using extensive simulations and real-world experiments over the Internet. To gain further insight into age control, we also empirically compare ACP with a mix of loss-based, delay-based, and hybrid congestion control algorithms used by TCP.

While, as expected, state-of-the-art TCP (hybrid) congestion control attempts to keep a number of bytes given by the product of the bottleneck rate and baseline RTT in the network pipe, the bottleneck rate and the baseline RTT may not shed as much light on the age optimizing rate of updates. *To exemplify from our experiments, when TCP sends segments over an end-to-end path consisting of a* 24 *Mbps* 802.11*a link followed by an intercontinental path over the Internet, it saturates the* 802.11*a link, which has the bottleneck rate for the path.* Age, however, is optimized at a much lower rate of about 1 Mbps. It turns out that the intercontinental path, much faster than the 802.11a link is, in fact, the constraining factor with respect to the achievable age over the end-to-end path, likely because of the other traffic flows that utilize the intercontinental path. We also observe that at the age optimal rate, depending on the network scenario, a source may send multiple updates per round-trip-time (RTT) or may send an update over many RTT. In general, the bottleneck link rate and the baseline (updates sent in a stop-and-wait manner) RTT may not shed light on the age optimal rate.

Age being optimized at low update rates has consequences with respect to multiple ACP end-to-end flows sharing wireless access to send updates to recipients in the cloud. We experimented with a large number of sources (up to 80 WiFi nodes in a high node density setting and a fixed physical layer rate of 6 Mbps, where each node was a source using ACP) sharing an access point to send their updates to a cloud server over the Internet. We show that ACP allows sources to share the access well. When the wireless access isn't the constraining factor, given the low age minimizing rate over the end-to-end path, all sources send at the minimizing rate. As the number of sources increases, the resulting congestion over the WiFi access has ACP gradually reduce the rate of updates per source in a manner such that the sources together fully utilize the WiFi access link rate. While fully utilizing the access like TCP, ACP, however, keeps age much lower than TCP congestion control algorithms. Even the packet retry rates because of collisions over WiFi are much lower than for TCP. Last but not the least, using controlled simulations, we show that ACP's sharing behavior is in line with what would be expected of a good age control strategy enabling sharing of access amongst multiple sources.

We conclude the thesis with a study on the co-existence of ACP and TCP flows

when the flows share an end-to-end Internet path over a shared WiFi access [14]. We found that ACP performance in co-existent flows can be improved by assigning a higher Differentiated Services Code Point (DSCP) priority to the ACP flows. However, the gains from prioritizing ACP flows vanish quickly with an increase in contention over the shared WiFi multiaccess. ACP is publicly available at [31].

1.1.3 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 proposes QAware, a novel cross-layer approach for MPTCP scheduling. We find that the MPTCP scheduling decision can be significantly improved by incorporating readily available local information from the device driver queues in the decision-making process. QAware uses this local queue buffer occupancy information along with the end-to-end delay estimate (RTT). We evaluate and compare QAware's performance to existing schedulers through simulations and real experiments. In Chapter 3, we describe our novel transport layer protocol, namely the Age Control Protocol (ACP) and ACP+, that enables timely delivery of IoT updates to monitors in a network-transparent manner. We detail the protocol and the proposed control algorithm. We demonstrate the efficacy of our proposed protocols using extensive simulations and real-world experiments constituting up to 80 clients and a server in the cloud. Next, in Chapter 4, we conduct an in-depth, real-world study on ageing of IoT updates over an end-to-end Internet path using a mix of loss-based, delay-based and hybrid TCP congestion control algorithms. Our evaluation considers a core network and contended and shared wireless access network. Lastly, we study the impact of prioritizing the age-sensitive traffic in the presence of coexisting throughput hungry traffic in Chapter 5. We find that prioritizing ACP+ has low gains *age-wise* in the presence of contention. We conclude this thesis and discuss future research directions in Chapter 6.

Chapter 2

QAware: A Cross-Layer Approach to MPTCP Scheduling

2.1 Introduction

The next-generation applications such as augmented reality (AR), virtual reality (VR), teleoperated driving and video streaming demand very high throughputs and low delays for their optimal operation. While improvements in communication technologies have increased data rates and reduced packet latencies, there are still many challenges. There have been attempts to cater to the QoS requirements of such emerging applications by designing improved TCP congestion control protocols that adapt to network [32, 33, 34], or by adding reliability and faster connection capability to UDP through QUIC [25, 26]. These improvements, however, aim to maximize the end-to-end performance over a single path. If the link characteristics over a path deteriorate, because of packet drops or excess queueing, there are little such solutions can do to improve the performance. To this end, Multipath TCP (MPTCP), standardized in early 2013, allows devices with multiple network interfaces, e.g., smartphones with WiFi and LTE, to seamlessly form multiple parallel connections to exploit the full network capacity. MPTCP offers increased robustness and resilience, as well as seamless handovers and it has been proposed to be also used in datacenters [35] and opportunistic networks [36]. Due to the performance benefits of MPTCP compared to TCP, several known organizations have incorporated the protocol within their products and services and its usage in the Internet has been steadily increasing [37, 38]. Apple uses MPTCP in its iOS devices to enhance the user experience surrounding its system services, e.g., Siri, Music, Maps, Wi-Fi Assist [39]. In 2019, Apple provided APIs to third-party developers for making use of MPTCP in non-system iOS applications. Korea Telecom, in partnership with Samsung, uses MPTCP to provide Gigabit speeds over Wi-Fi and LTE to its customers [40]. In Febru-

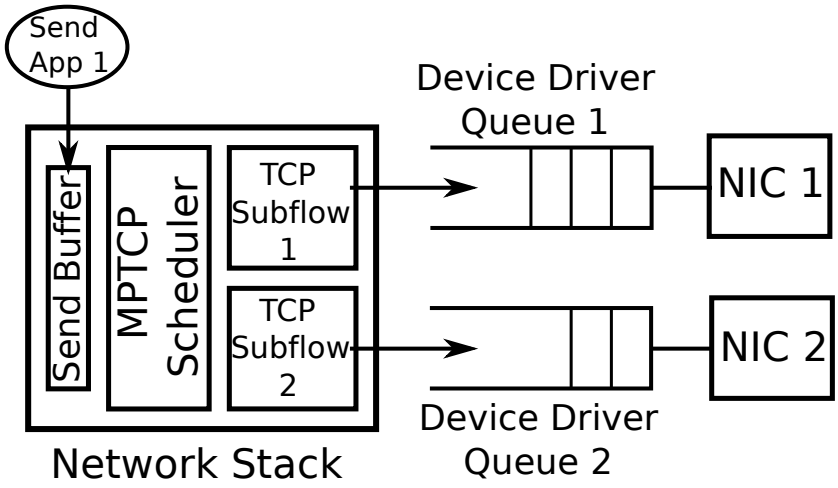


Figure 2.1: An illustration of MPTCP-compliant machine and how its subflows interact with their corresponding network interface queues.

ary 2020, MPTCPv1 was upstreamed to Linux and is now available to all users running Linux 5.6 or newer [41].

Figure 2.1 shows the network stack of MPTCP-compliant machine. Applications utilizing MPTCP can send their data over multiple TCP subflows, where each subflow is associated with a unique network interface. TCP packets scheduled over a subflow wait in the device driver queue of the corresponding network interface before transmitting them by the network interface card (NIC). The choice of network path for sending application data is made by the MPTCP scheduler block and depends on the scheduling policy.

Scheduling between the multiple connections is an obvious research problem and recently multiple proposals [42], [43], [44], [45] have emerged to improve the default minSRTT MPTCP scheduler [21]. Typically, these schedulers use a transport layer estimate of the end-to-end bandwidth/delay (for example, the smoothed round-trip time) for each TCP subflow as an input to the scheduling policy that decides how the application data must be assigned to the multiple subflows. However, we found that as a particular flow is used more, its end-to-end delay increases gradually, making it less attractive. But the traditional, purely end-to-end-based estimation reacts very slowly to these changes.

This observation acts as our main motivation to design a novel scheduler for MPTCP, QAware, which departs from the previous scheduling proposals in a fundamental way. While we also use the end-to-end delay estimates, like current

schedulers, QAware additionally considers the number of packets in the device driver queue of the sender. This modification is motivated by our findings, which we discuss further in Section 2.3. Additionally, utilizing queue occupancy information allows QAware to use all available subflows optimally, especially when their properties are highly heterogeneous. Existing proposals like [42, 45, 46], treat the flows as separate entities and typically do not fully use all the flows. QAware optimizes transmission over all the flows and gets a significantly higher aggregate throughput, with no loss of performance in any situation.

The key contributions summarized in this chapter are:

1. We propose QAware, a novel cross-layer approach to scheduling packets across all available MPTCP subflows. The design is motivated by our experimental findings; combining local device driver queue occupancy with the traditional end-to-end delay measurements yields far superior throughput performance.

2. We model available MPTCP subflows as multiple parallel service facilities that can service data provided by an application. This enables us to leverage queueing theoretical insights to create a scheduling policy that combines end-to-end delays and device driver queue occupancy.

3. Our simulations and real-world experimentation over a wide range of applications compare QAware with the default MPTCP scheduler – minimum SRTT (minSRTT) [21], Earliest Completion First (ECF) [45], Delay Aware Packet Scheduler (DAPS) [44], and Blocking Estimation-based scheduler (BLEST) [42].

4. We have implemented QAware in MPTCP v0.93 and have made our code available [24].

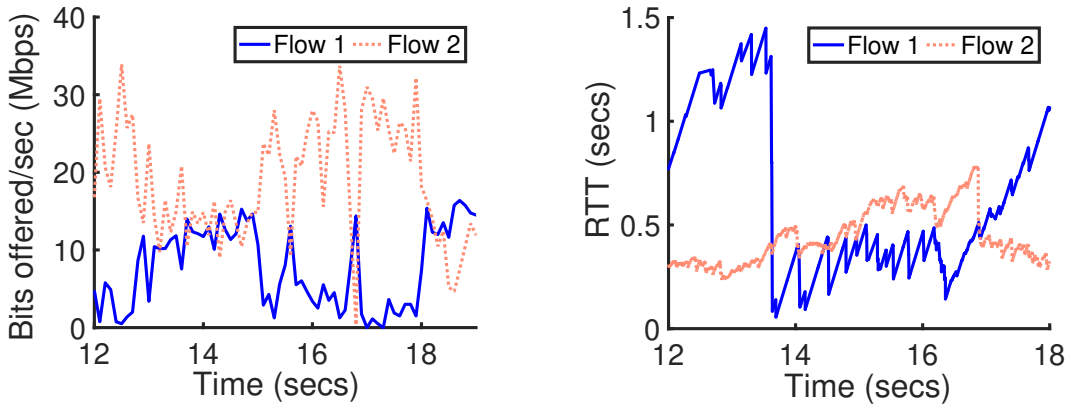
2.2 Related Work

The default MPTCP scheduler (minSRTT) allocates traffic on the fastest subflow (one with the smallest smoothed RTT) with available congestion window at each packet arrival. Several researchers have proposed improvements to the default

minSRTT scheduler. Most approaches leverage the difference in RTT of the subflows [47, 48]. Others have also considered additional TCP-layer parameters such as SSThresh, congestion window, selective `ACK` and receiver buffer size along with RTT [49, 50, 51].

In [46], the authors introduce an additional sender queue to schedule packets on a subflow even when it is unavailable. Delay Aware Packet Scheduler (DAPS) [44] generates a schedule for sending future segments over subflows based on their RTT ratios. However, this makes DAPS unable to react promptly to network changes due to pre-computed long schedules. Blocking-Estimation-based MPTCP Scheduler (BLEST) [42] aims to reduce head-of-line blocking by waiting for the faster subflow despite the space availability in the congestion window of the slower subflow. ECF [45] follows a similar principle as that of BLEST, but while BLEST aims to reduce out-of-order delivery, assuming that the send buffer is a bottleneck, ECF aims to minimize completion time.

Researchers have also proposed schedulers that improve MPTCP performance for specific application use-cases. Decoupled Multipath Scheduler (DEMS) [43] aims to reduce fixed-size file delivery time over MPTCP by estimating available bandwidth on subflows. However, the authors rely on exact knowledge of data chunk boundary for efficient scheduling. In [52], authors leverage application layer information for flow scheduling decisions to provide delay-resilient video streaming in MPTCP. MP-DASH [53] exploits path information from streaming client to improve DASH video delivery. [54] labels WiFi subflow as active/inactive for data transmission based on a minimum desired signal strength. However, unlike other cross-layer approaches which optimize specific application performance over MPTCP, QAware taps into lower layer information to improve performance for *all* MPTCP traffic. Furthermore, as shown later in the chapter, QAware's unique design of leveraging hardware queue occupancy enables it to swiftly adapt to varying network conditions and co-existent network applications sharing bottleneck paths.



(a) Loading (Mbps) of subflows

(b) RTT(s) of subflows

Figure 2.2: Loading (Mbps) and RTT(s) of the subflows. The paths taken by the subflows and the network are shown in Figure 2.3.

2.3 Motivating Use of Cross-Layer Information

Figures 2.2a and 2.2b respectively show loading (bits offered per second) and the corresponding estimates of round-trip times (RTT) of two available subflows by the default MPTCP scheduler, minSRTT. They were obtained from controlled testbed experiments and show how the scheduler optimizes over two TCP subflows using non-interfering end-to-end paths. The network topology used in the experiment is shown in Figure 2.3. The last-mile links were WiFi using 802.11g, and the rest were 1 Gbps Ethernet. Neither flow dropped any packets during the length of the experiment.

In the experiment, the default scheduler only utilizes $\approx 60\%$ of available aggregated bandwidth. Observe (Figure 2.2a) that the default scheduler, more often than not, prefers to send packets on one flow over the other. However, this by itself is not responsible for the low utilization of the available bandwidth. We argue that the default scheduler loads a flow deemed to be the best amongst available flows for *undesirably long* intervals. This is because the scheduler uses only the SRTT of the flows, which is a delayed end-to-end transport layer measurement, for its scheduling decisions.

Consider the RTT of flow 1 in Figure 2.2b. The RTT captures in a lagged manner the impact of scheduling decisions on the subflow. The consistently high values (see interval 12s to 14s in the figure) correspond to an earlier interval of time

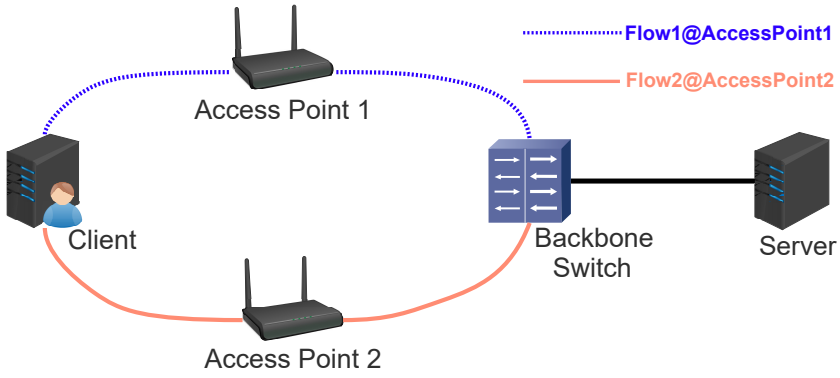


Figure 2.3: Topology used in experiments and simulations.

when the subflow was being assigned packets by the scheduler while it was heavily loaded. That is, the device queue corresponding to the subflow had previously many packets queued at the NIC.

The sharp dip in values (around time 14s in the figure) captures the transition from when the flow stopped being assigned packets due to high RTT to when it was again assigned packets. These assigned packets arrive at a rather lightly loaded flow and see much smaller RTT, which causes the dip. The small RTT that follows the dip corresponds to packets being assigned to the flow while it was still lightly loaded. As the subflow continues to be assigned packets, the same is reflected, albeit in a delayed manner, in increasing RTT (seconds 16 to 18 in Figure 2.2b) that eventually peaks as it did during $12 - 14$ seconds. By the time the resulting large RTT makes the scheduler switch to the other flow, the scheduler has already spent an undesirably long time injecting packets to a loaded subflow.

In summary, the scheduling decisions that led to high device queue occupancy and an increase in RTT were made using values of RTT that corresponded to an earlier interval when the flow was less loaded. So while a device queue (local to the MPTCP sender and used by the MPTCP flow) is loaded with packets, MPTCP scheduler remains oblivious to the same. Instead, it waits to be informed via a delayed end-to-end RTT-based feedback mechanism. In the process, it loses out on many opportunities of scheduling packets to the other better flow, one that is lightly loaded.

The above observations motivate QAware. It uses the device queues' occupancy and RTT estimates to use all available flows more efficiently.

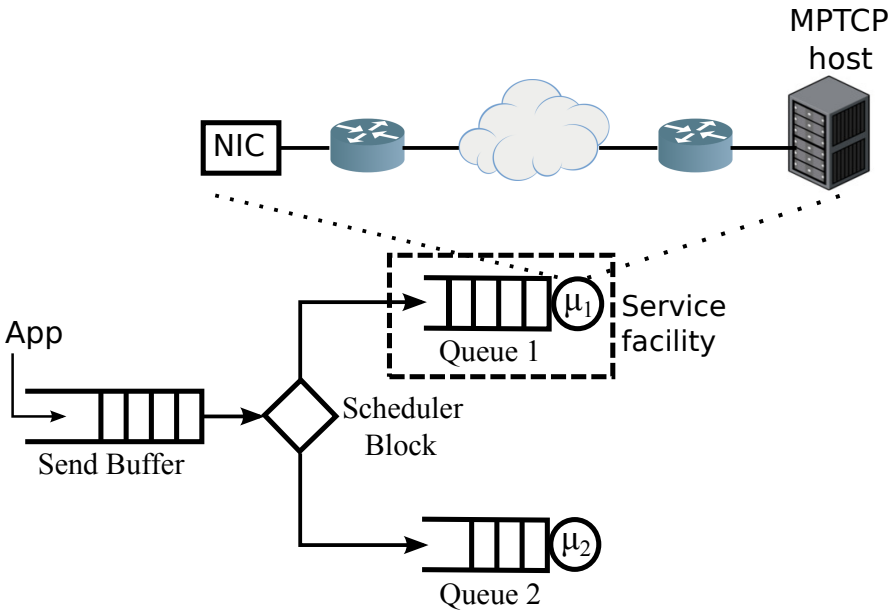


Figure 2.4: Queueing abstraction of an *end-to-end* MPTCP connection with two subflows.

2.4 QAware Scheduler

We consider a simplified queue-theoretic abstraction to capture the essentials of the scheduling problem, with the goal of maximizing *end-to-end* throughput. Specifically, we model each subflow by a service facility. Figure 2.4 illustrates the abstraction for an MPTCP *end-to-end* connection that uses two TCP flows. The abstraction allows us to apply results from analysis of multi-queue systems[55].

In our queueing abstraction, packets generated by an application arrive in a queue that models the TCP send buffer (Figure 2.1). Packets in this queue are assigned to one of the available service facilities in a first-come-first-serve (FCFS) manner. Each facility consists of a finite queue and a server. Packets inside a facility are serviced in an FCFS manner. The queue in a service facility is the device driver queue (Figure 2.1) that is used by the TCP subflow corresponding to the facility. The server includes the source host NIC, access network used by the subflow, intermittent nodes in the core and the destination host (all layers of the TCP/IP stack). When a packet is assigned to a service facility, it may find other packets waiting for service in the facility's queue. This packet must wait for all the other waiting packets to finish service before it enters the server of the facility. The total time a packet spends in a facility, often referred to as its *system time*,

includes the time it waits in the facility's queue and the time it spends getting service.

2.4.1 Origins of the QAware scheduler

Many analytical works on queueing systems have looked at scheduling customer/-packet arrivals to parallel service facilities [55, 56, 57, 58]. For many general arrival processes and service time distributions, when all servers are stochastically identical, the optimal policy is to choose a service facility with the minimum number of packets in its queue [55, 56, 58], that is it minimizes the average packet system time. For the case of non-identical servers, a scheduling policy that assigns a packet to a service facility that minimizes the conditionally expected system time of the packet, conditioned on the knowledge of the number of packets waiting for service in the facility, shows good performance [55]. Our QAware scheduler uses the policy in MPTCP setting. Previous research has demonstrated that considering low-layer queue occupancy can improve MPTCP congestion control performance [59].

Consider K service facilities indexed $1, \ldots, K$. Let facility k have a service rate of μ_k. The two facilities in Figure 2.4 have service rates of μ_1 and μ_2. Let $n_k(t)$ be the number of packets waiting for service in facility k at time t. The policy assigns a packet to a service facility k^* given by

$$k^* = \arg\min_k \frac{n_k(t) + 1}{\mu_k}. \tag{2.1}$$

Note that $1/\mu_k$ is the expected service time of a packet in facility k. Thus, the conditional waiting time of a packet that enters such a facility is $n_k(t)/\mu_k$, which is the sum of the expected service times of the $n_k(t)$ packets currently waiting for service in the facility. In addition, we add the term $1/\mu_k$ to $n_k(t)/\mu_k$, to include the expected service time of the packet to be scheduled. Thus, the expression being minimized in (2.1) is the conditional expected *system time* of a packet if it were to be assigned to facility k.

2.4.2 Adapting scheduling policy to multiple end-to-end TCP subflows

The number $n_k(t)$ of packets in the queue of service facility k is the number of packets waiting in the device driver queue of the corresponding subflow k and can be obtained. However, we must estimate the average service time $1/\mu_k$ of subflow k.

Consider the i^{th} packet arrival. Let t_i^s be the time the packet is assigned to a subflow. Let t_i^a be the time that a TCP `ACK` acknowledges receipt of the packet. The round-trip time of the packet is $\text{RTT}_i = t_i^a - t_i^s$. Note that this includes the time packet waits in the device driver queue of its assigned subflow before it starts service and the time it spends in service. This is the *system time* of the packet. Let W_i[1] be the time the packet i waits in the queue. This time can be calculated locally at the MPTCP sender. The time X_i that the packet spends in service begins when the packet enters the NIC for transmission and ends when a TCP `ACK` for the packet is received. Given W_i and RTT_i, we have $X_i = \text{RTT}_i - W_i$. The estimate of the service time is updated on receipt of a TCP `ACK`. Let $\hat{S}_k$ be the current estimate of the average service time of facility k. On receipt of a TCP `ACK` for packet i, we update

$$\hat{S}_k = \alpha \hat{S}_k + (1-\alpha) X_i, \tag{2.2}$$

where $0 < \alpha < 1$ applies appropriate weights to the last estimate of the average and the current service time. We use $\alpha = 0.8$ in this work which is also the smoothing factor for TCP congestion control [2] . The corresponding estimate of the service rate is $1/\hat{S}_k$. At time t, QAware schedules to the TCP subflow k^* that satisfies

$$k^* = \arg\min_k (n_k(t) + 1)\hat{S}_k. \tag{2.3}$$

Finally, note that since $X_i = \text{RTT}_i - W_i$, we have $\hat{S}_k = \text{RTT} - \widehat{W}$, where RTT

[1]For simplicity of exposition we ignore the time a TCP `ACK` may have to wait in a queue before being sent to the TCP layer.

[2]We examined for other values of α. This did not impact the overall performance of QAware.

Algorithm 1 QAware Algorithm

Available Subflows `SF`$\in \{1, \ldots, n\}$
`minService` $\leftarrow$ `0xFFFFFFFF`
`selectedSubflow` $\leftarrow$ NONE
/*The function below will return best subflow for packet P_k*/
for each `subflow` $\in$`SF` **do**
  $n_k \leftarrow$ `queueSize(subflow)`
  if $n_k \neq 0$ **then**
    $\Delta t \leftarrow$ sampling time
    Δpackets $\leftarrow$ packets dequeued in Δt
    $W_k \leftarrow [1/(\frac{\Delta \text{packets}}{\Delta t})]n_k$
  else
    $W_k \leftarrow 0$
  end if
  $\widehat{W} \leftarrow \alpha\widehat{W} + (1-\alpha)W_k$
  $\hat{S}_k = [\text{RTT} - \widehat{W}]$
  $TS_k = (n_k + 1)\hat{S}_k$
  if $TS_k <$ `minService` **then**
    `minService` $\leftarrow TS_k$
    `selectedSubflow` $\leftarrow$ `subflow`
  end if
end for

and $\widehat{W}$ are the exponentially weighted moving averages, with coefficient α, of packet round-trip times and device driver queue waiting times, respectively, for the subflow k. In our real implementation, summarized in (Algorithm 1), we use RTT estimates that are readily available for each subflow and we calculate an approximation of $\widehat{W}$ based on information available from device driver queues.

2.5 Implementation

We implement QAware as a modular scheduler using MPTCP v0.93 based on Linux kernel v4.9.60 [60]. The code is available at [24].

As shown in Section 2.4, QAware's functioning depends on the current estimate of network interface (NIC) queue occupancy. Conventionally, the NIC queues were either implemented within the hardware itself or as part of the driver, which made NIC queues invisible to the Kernel and its occupancy extremely hard to estimate. However, since Linux Kernel > v3.3.0, several NIC queue management protocols, known as Byte Queue Limits (BQL), have been introduced as part of the Kernel code to resolve starvation and latency at the NIC [61]. The BQL algorithms push

queueing abstractions from hardware drivers to specific data structures, which can be accessed from within the Linux kernel[3].

Our implementation closely follows the Algorithm 1. We first tap the network device address mapped to MPTCP socket via `struct dst_entry` to access *DQL*[4] as follows:

```
dql = netdev_get_tx_queue(dst->dev)->dql
```

We further utilize *DQL* entry to estimate current NIC (*netdevice*) queue occupancy of each MPTCP subflow.

$$\texttt{qSize} = \{\texttt{dql->num\_queued - dql->num\_completed}\}$$

Here, `num_queued` and `num_completed` refer to the total number of bytes queued in the network device and the number of bytes successfully transmitted by the device, respectively.

Apart from NIC queue estimates, we utilize the smoothed mean RTT estimates in microseconds via `srtt_us` accessible through `struct tcp_sock`. We ensure that our implementation is in line with guidelines mentioned in RFC 6182 [19].

2.6 Evaluation Methodology

We evaluate QAware's performance through extensive simulations and real-world experiments in the following sections. We model our evaluation methodology to mimic real MPTCP network configurations and application use-cases. In the majority of our evaluation, we model a realistic network scenario (as illustrated in Figure 2.3) wherein a client leverages two distinct network paths to connect to a distant server.

For simulations, we implement QAware on the ns-3 network simulator. We compare QAware with default minSRTT, and Earliest Completion First (ECF) [45]

[3]At the time of QAware's implementation and experimentation, only PCIe-based ethernet drivers supported BQL [62].

[4]In Linux, BQL is implemented as Dynamic Queue Limit (DQL).

scheduler for constant bit rate (CBR), file downloads, and web browsing workloads. The simulations help us zoom into the workings of the schedulers and evaluate QAware over various workloads and network path configurations. Our evaluation setup and results are described in Section 2.7.

We further examine and validate the performance gains obtained by QAware in simulated environments via real network experiments. We utilize our Kernel implementation summarized in Section 2.5. The experiments were performed in a university data center and considered a variety of workloads such as video streaming, web file downloads, etc. We compare QAware with several state-of-the-art schedulers such as minSRTT, Delay Aware Packet Scheduler (DAPS) [44], Blocking Estimation based scheduler (BLEST) [42], and ECF [45]. The details of our experiments and consequent results are discussed in Section 2.8. All our results are averaged over multiple runs.

2.7 Simulation Setup and Results

We simulated network topologies of the kind shown in Figure 2.3. For all simulations, the links between the access points and the backbone switch and between the backbone switch and the server were modeled as wired links with rates 30 Mbps and 50 Mbps, respectively. The client is connected to the two access points over wireless links with physical layer (PHY) rates in the range $6 - 12$ Mbps. These two wireless links provided the two network paths to send application data. Both subflows use independent congestion control.

We simulated the following *applications*: i) constant bit rate (CBR) data from low to high rates, ii) file transfer for sizes of $10 - 30$ MB, iii) web browsing of top 10 out of the US Alexa-100 websites, and iv) CBR with one of the paths being shared by UDP traffic. For the applications, we simulated the following *network configurations*: i) both wireless links have the same rate, ii) one link is much faster than the other, and iii) one link drops TCP packets.

Comparisons of QAware with ECF and minSRTT[5] demonstrate the benefits

[5]In the simulation, the scheduler assigns packets over independent TCP streams. We do not incorporate other MPTCP functionality such as re-transmission handler and path manager.

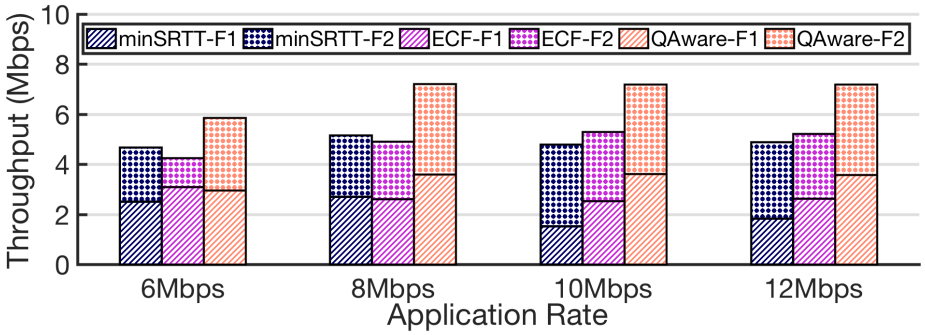


Figure 2.5: Subflows F1 and F2 use links with PHY rates of 6 Mbps.

that QAware accrues because it optimally utilizes both network paths.

2.7.1 Constant Bit Rate Traffic

Access paths with no packet errors. Figure 2.5 shows the TCP throughputs obtained by the schedulers for increasing CBR rates. Each wireless link was configured with a PHY rate of 6 Mbps. This results in homogeneous network paths. On average, QAware achieves percentage throughput gains of about 40% over the rest. Further, note that all schedulers use both subflows. However, unlike the others, QAware utilizes both the subflows almost equally for the entire simulation time for all the CBR loads. To better understand their behaviors, consider Figure 2.6, Figure 2.7 and Figure 2.8, which shows the variation of throughput, device driver queue occupancy, and smoothed RTT, as a function of time, for a 2 second interval for minSRTT, ECF and QAware scheduler respectively. The CBR rate was set at 12 Mbps. From the subflow throughputs and queue occupancy, it is clear that QAware uses both subflows almost simultaneously. ECF uses just one subflow for most of the interval, and while minSRTT uses both flows during the interval, it switches between them very infrequently. Both minSRTT and ECF rely on the delayed feedback provided by SRTT and so end up scheduling packets to one subflow for longer intervals than QAware. Essentially, they switch flows when SRTT of the subflow in use exceeds that of the other subflow. In addition, ECF, by design, declines scheduling opportunities to a subflow with a larger RTT and prefers to wait for faster subflows. This explains the reason for using one flow for a longer duration than minSRTT scheduler. In minSRTT and ECF,

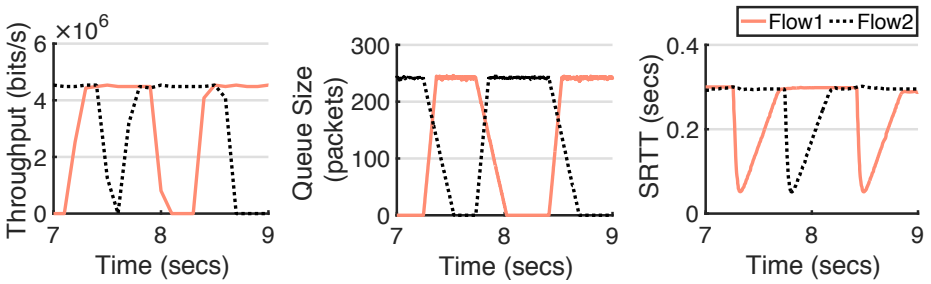


Figure 2.6: minSRTT Scheduler

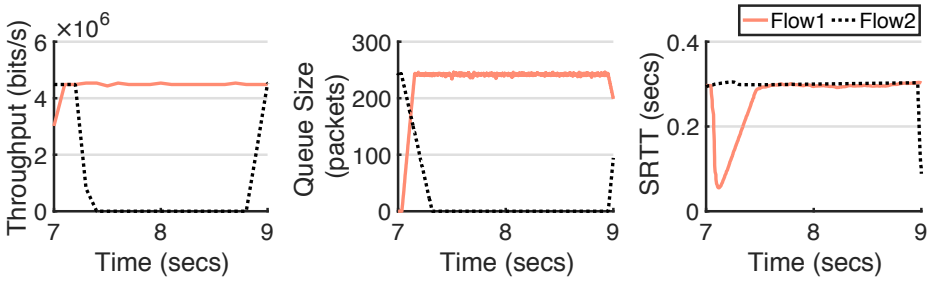


Figure 2.7: ECF Scheduler

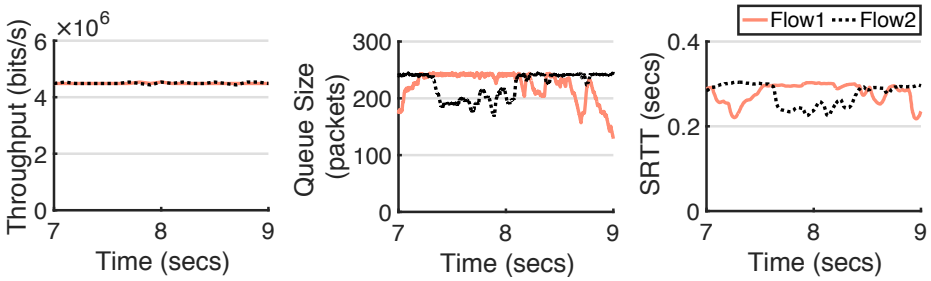


Figure 2.8: QAware Scheduler

subflows experience swings in SRTT. The SRTT increases linearly while it is the subflow of choice. This increase eventually makes the subflow less desirable than the other and the scheduler switches to the other flow, which, due to the current low occupancy in the corresponding device queue, experiences low SRTT.[6]

Figure 2.9 shows throughputs obtained by the CBR application when the PHY rate of one of the wireless links is 6 Mbps and the other is 12 Mbps. While all schedulers equally utilize the subflow using the 12 Mbps link, QAware also utilizes the subflow mapped on the 6 Mbps link. On average, QAware achieves throughput gains of about 50% over the rest.

Access paths with packet errors. Figure 2.10 shows the throughput obtained when one subflow suffers a packet loss rate of about 10^{-2}. Both wireless

[6]Our observations with respect to QAware and minSRTT for three homogeneous paths are similar.

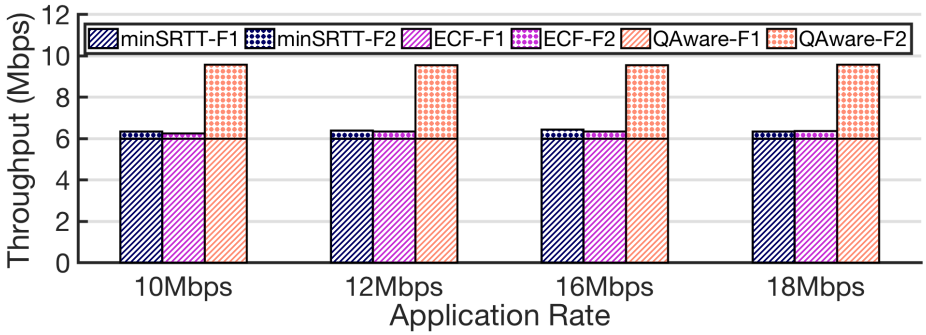


Figure 2.9: Subflow F1 and F2 use links with PHY rate of 12 Mbps and 6 Mbps respectively.

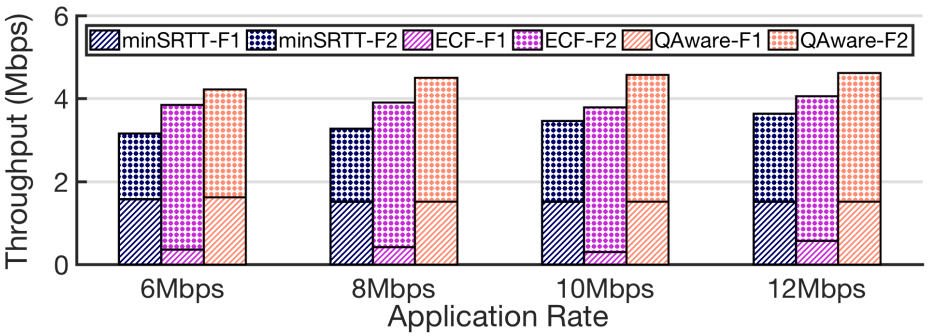


Figure 2.10: Per-flow throughput comparison for different CBR rates where subflow F1 experiences a packet drop rate of 10^{-2}.

links have PHY rates of 6 Mbps. Upon detecting packet loss, the congestion window of the subflow decreases based on *TCP congestion avoidance algorithm*, which limits the number of packets that can be sent on that subflow. Even in this situation, QAware is able to exploit both subflows better and achieves about 32% and 15% improvement over minSRTT and ECF respectively. When the wireless links are 12 Mbps and 6 Mbps with an error on the slower link, the corresponding gains are 53% and 6% (figure not shown due to space limitations). Since ECF is biased toward using the faster path, it performs almost as well as QAware when the error-free path has a faster wireless link. On the other hand, while minSRTT uses the error-prone path better than ECF, it is unable to make good use of the error-free path as the other two schedulers.

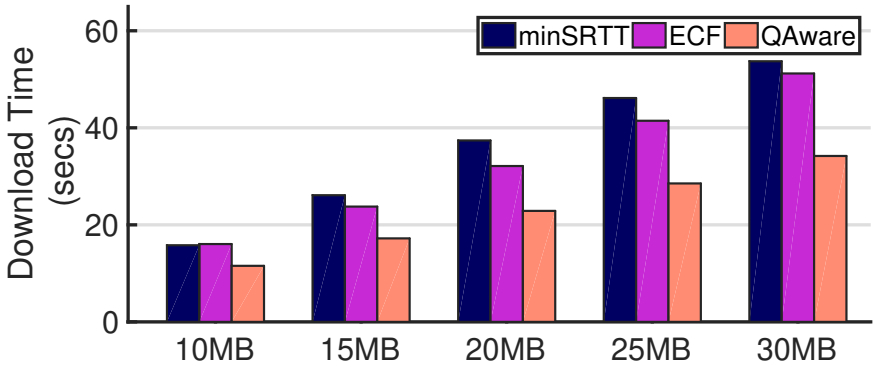


Figure 2.11: File download completion times when both subflows use wireless link with PHY rate of 6 Mbps.

2.7.2 Fixed Size File Transfer

Figure 2.11 shows the download completion time achieved by the three schedulers for five different file sizes ranging from 10MB to 30MB. Both wireless links were set to a PHY rate of 6 Mbps. Observe that QAware obtains the least download time for all the file sizes. This is explained by its ability to utilize both the subflows for data transfer effectively. The performance gap increases proportionally with file size. Overall, QAware achieves 35% and 30% reduction in average download time over minSRTT and ECF, respectively.

2.7.3 Web-browsing

To simulate web browsing, we deployed objects of 10 of the top U.S. Alexa-100 websites, summarized in Table 2.1, in our simulated server. The client consecutively downloaded relevant objects of each website from the server at a variable rate between 10Mbps to 30Mbps chosen in a probabilistic manner. We compared scheduler performance for when both wireless links are 6 Mbps and when one of the

Website	News	Tech	Radio	Shopping	Finance
#Objects	202	67	66.2	52.2	39.7
Size (KB)	3821.2	2152.2	2453	1000.7	1988.1
Website	Wiki	Market	Social	Movie	Travel
#Object	28	49	69	39	21
Size (KB)	601.2	2032.8	1700.2	845.7	2000.4

Table 2.1: Web objects for traffic generation

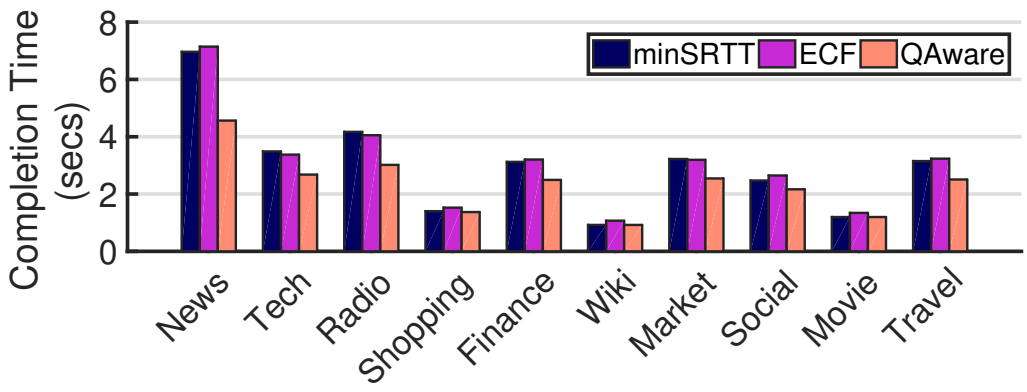


Figure 2.12: Download completion time for 10 websites from top U.S. Alexa-100 websites.

links is 12 Mbps. QAware achieves a significant reduction in download completion time for both configurations, specifically up to 35% for the former (see Figure 2.12) and up to 28% for the latter (figure not shown due to space limitations). On the other hand, ECF and minSRTT perform similarly.

2.7.4 Multiple Applications

In current computing environments, end hosts typically run multiple applications which must share the interfaces available at the host for network transfers. An ideal MPTCP scheduler must efficiently adapt to bandwidth competition on bottleneck links in such coexisting environment. We used the following setup to evaluate the impact of such sharing on the schedulers. The PHY rates of the wireless links were set to 9 and 6 Mbps. A CBR application generated data for a 10-second interval and used both the MPTCP subflows. The results are shown in Figure 2.13.

Starting at 4 seconds, we introduced traffic from a UDP application that used the network path with the 9 Mbps wireless link. The greyed area in the figure denotes the time duration when both MPTCP and UDP applications were active at the client. The UDP traffic lasted for 4 seconds. Before the start of the UDP traffic, only QAware scheduler was utilizing both available subflows. Once the UDP application starts, the device queue of the 9 Mbps wireless link saturates. QAware, however, quickly adapts to it and reduces the traffic being sent on the corresponding subflow. All the while, it keeps utilizing the subflow over the slower wireless link. On the other hand, both minSRTT and ECF need to wait for several

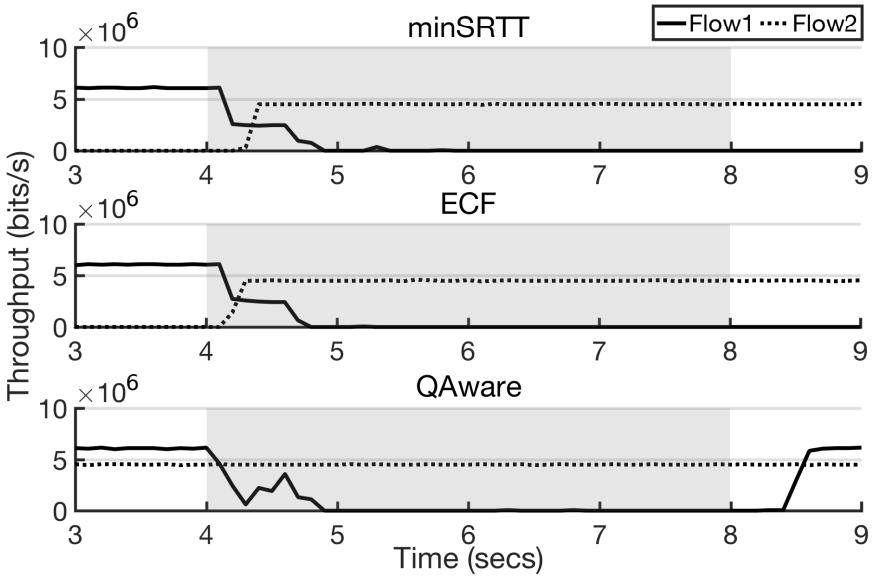


Figure 2.13: Per-flow throughputs when the interface used by subflow F1 sees UDP traffic for 4 seconds (greyed).

RTT updates for the impact of UDP traffic on queue wait times to get reflected in the SRTT of the subflow. Lastly, unlike the other schedulers, QAware is also quick to detect the availability of the subflow after the 8-second mark, which is when the UDP application stops its transfer. Overall, QAware leads to gains of about 40% over minSRTT and about 50% over ECF.

2.8 Real World Setup and Results

We next examine QAware's performance in real network environments. Figure 2.14 shows our test network topology a University data center in Finland. We assign two similar machines with 16 core AMD Opteron processors, 8 GB DDR2 RAM running Ubuntu 16.04 LTS with latest stable MPTCP implementation (version 0.93, based on Linux kernel v4.9.60 [60]) as client and server. The implementation uses the default congestion control algorithm (coupled OLIA). Both machines are interconnected via two separate Gigabit Ethernet interfaces. One Ethernet connection is routed through the internal University of Helsinki network and therefore encounters background traffic from University staff. It has an end-to-end RTT of >1ms. The other connection is over Top-of-Rack (ToR) switch with RTT <1ms.

We compare QAware with the following schedulers: i) minSRTT, ii) Delay Aware Packet Scheduler (DAPS) [44] iii) Blocking Estimation based Scheduler

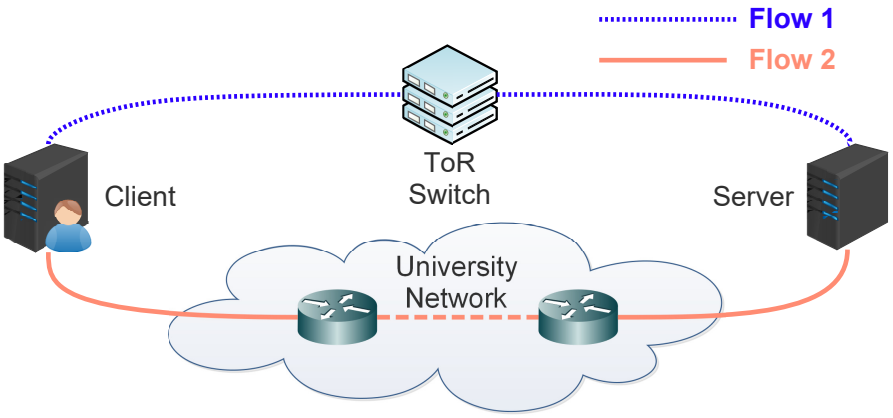


Figure 2.14: Real network testbed in university datacenter.

Delay (ms)	1	40	80	160
Bandwidth (Mbps)	950	600	300	200

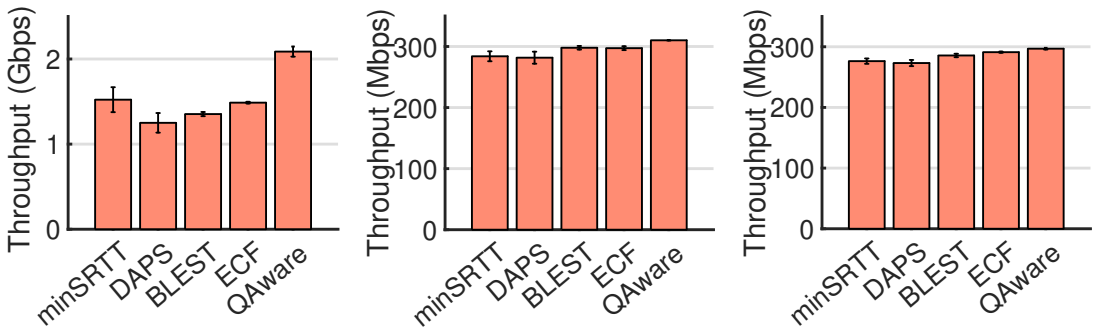
Table 2.2: Configurations for Bulk Traffic Experiments

(BLEST) [42], and iv) Earliest Completion First (ECF) [45] [7]. We first compare scheduler performance for an application generating bulk traffic. This workload provides a qualitative validation of the results obtained in Section 2.7. We further present scheduler performance for DASH video streaming and web file downloads. We used the Linux Traffic Control system (*tc*) in combination with a Hierarchical Token Bucket (HTB) packet scheduler using Statistical Fair Queuing (SFQ) for network shaping. We flushed out the TCP cache between runs to ensure that each run is independent of the next. All our results are averaged over ten runs.

2.8.1 Bulk Traffic

In this section, we compare QAware's performance with other schedulers for a high application transfer rate over both subflows. We performed experiments with different settings of delays along the two paths. The setting includes i) default path delays ($<$ 1ms and $>$ 1ms), ii) delay shaping to introduce 40ms of delay along one path and 80ms along the other, and iii) 40ms along one path and 160ms along the other. Path bandwidths corresponding to the different delays are stated in Table 2.2.

[7] DAPS, BLEST, and ECF are implemented on MPTCP v0.89 whereas the default minSRTT and QAware are based on MPTCP v0.93. We could not implement QAware on MPTCP v0.89 as it is based on Linux v3.18 which does not support BQL. Please see [60] for exact changes between the two versions.

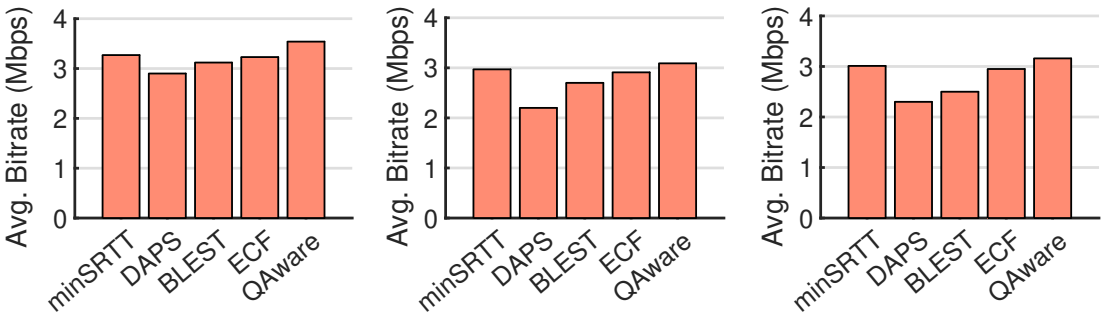


(a) Default (b) 40+80ms (c) 40+160ms

Figure 2.15: Bulk Traffic throughputs for different access path delays.

Figure 2.15a compares average throughput obtained by different schedulers for default path delays. QAware achieves more than 45% increase in throughput compared to DAPS, BLEST and ECF. QAware also provides an improvement of 37% over the default minSRTT scheduler. Interestingly, the minSRTT scheduler outperforms DAPS, BLEST, and ECF in the experiment. We attribute minSRTT's efficiency to two reasons. Firstly, DAPS, BLEST and ECF schedulers have been designed to improve MPTCP performance for heterogeneous delays along available network paths. In fact, BLEST and ECF go as far as not sending an available packet on a slower subflow and waiting for the faster subflow to become available. When subflows witness similar delays (as in the current case), the default scheduler places more packets on each path than DAPS, BLEST, and ECF. Secondly, based on the more recent MPTCP kernel, minSRTT enjoys several code improvements and optimizations.

For when the path delays are 40 and 80ms, QAware yields an average throughput of 310 Mbps which is an improvement of about 10% over the default scheduler and DAPS and 5% over ECF and BLEST (shown in Figure 2.15b). As presented in Figure 2.15c, all schedulers perform quite similarly to each other as all try to fully utilize the lower delay subflow when path delays are 40 and 160ms. In this case, QAware still manages to achieve an improvement of about 7% over the default scheduler and DAPS, and about 4% over BLEST and ECF.



(a) 2+2 Mbps (b) 2+1.6 Mbps (c) 2.4 +1.6 Mbps

Figure 2.16: Average bitrate in video streaming for different path bandwidths.

Bandwidth (Mbps)	2.4	2	1.6
Delay (ms)	10	20	30

Table 2.3: Configurations for Video Streaming Experiments

2.8.2 Video Streaming

Streaming is a dominant Internet use case and is widely adopted by content providers such as Netflix and YouTube [63]. We set up a DASH server and host *Big Buck Bunny*, available from a public dataset, on it [64]. We configured the streaming server to provide five representations of the video from 240p to 1080p (same as most content providers). We re-encoded each representation in at least three different bitrates with overall available bit rates from 128Kbps to 3.8Mbps using H.264/MPEG-4 AVC codec. The streaming client employs an Adaptive Bit Rate (ABR) algorithm to download video segments according to the available network bandwidth. We throttled our testbed bandwidth to match the bitrates of DASH encodings. Table 2.3 shows the average delay measured at client-side for each bandwidth configuration. We evaluate and compare QAware's performance with other schedulers for when the two subflows i) have bandwidths of 2 Mbps, ii) have bandwidths of 2 Mbps and 1.6 Mbps, and iii) have bandwidths of 2.4 Mbps and 1.6 Mbps.

From Figure 2.16, we observe that QAware improves the performance of streaming applications in all network conditions. The performance improvement is more significant in scenarios where the path bandwidths are similar (8% and 5% with respect to default and 10% and 6% with respect to ECF, in Figures 2.16a and 2.16b respectively) as QAware utilizes available paths more efficiently than other

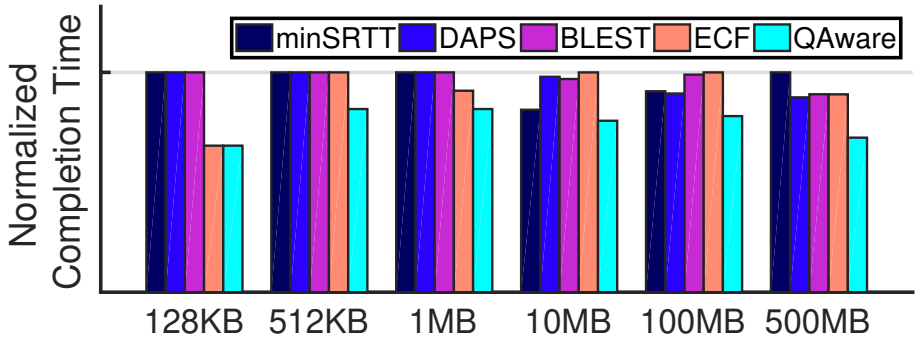


Figure 2.17: Normalized download completion time for different file sizes (smaller is better).

schedulers. DAPS consistently gives the worst performance out of all schedulers due to its strong dependence on the RTT ratio of two subflows.

2.8.3 Web File Download

We now evaluate QAware's performance for simple web downloads using *curl*. We set up an HTTP server using Apache 2.2.22 and hosted varying file sizes of range 128KB to 500MB. We eliminate application connection time by only considering the transport-level time in the overall download completion time observed at the client. Figure 2.17 presents the average completion time normalized to the maximum achieved value by the scheduler for given file size.

For small web transfers (<1MB) all schedulers perform quite similar to each other (it took 0.002s to download a 128 KB file by QAware vs. 0.003s by minSRTT). This is because for small data transfers, the bandwidth of the primary subflow is more than capable of single-shot transmission, and thus, MPTCP rarely switches to the secondary subflow. Therefore, until the performance of the primary subflow degrades during transfer, the choice of the scheduler does not affect the performance for small files. The default and DAPS scheduler achieve lower completion times for medium file sizes (≈10/100 MB) in comparison to BLEST and ECF. This is likely because BLEST and ECF add additional delays by waiting for the faster subflow to become available. For large files (500 MB), BLEST and ECF utilize faster subflow more efficiently than default and DAPS, thus achieving a lower completion time. QAware consistently outperforms other schedulers and

realizes up to a 20% decrease in completion time for medium file sizes (0.709s by QAware vs. 0.895s by ECF for 100 MB file) and 30% for large file downloads (3.46s by QAware vs. 4.93s by minSRTT for 500 MB).

2.9 Chapter Summary

We design QAware, a novel cross-layer MPTCP scheduler that combines hardware device queue occupancy and TCP RTT for efficient scheduling decisions. We evaluated QAware using extensive simulations and real network experiments for various network configurations and applications such as bulk data transfers, web browsing, web file downloads, and video streaming. Comparisons with various state-of-the-art schedulers such as DAPS, BLEST, and ECF were used to demonstrate the efficacy of QAware. It outperformed other schedulers in all network configurations and workloads we tested. Further, we show that QAware quickly adapts to co-existing applications and sudden variations in network conditions. We have open-sourced QAware's implementation as a modular scheduler for MPTCP v0.93 Linux release.

Chapter 3

Age Control Protocol: An End-to-End Transport Protocol Provisioning Freshness Over the Internet

3.1 Introduction

The availability of inexpensive embedded devices with the ability to sense and communicate has led to the proliferation of a relatively new class of real-time monitoring applications such as health care, smart homes, transportation, and natural environment monitoring. Figure 3.1 shows the typical end-to-end connectivity of such applications. IoT devices are deployed alongside users in home/offices/cities and connect to the network via a gateway over a wireless last-mile (for example, WiFi). Such devices repeatedly sense various physical attributes of a region of interest, for example, traffic flow at an intersection. This results in a device (the *source*) generating a sequence of packets (*updates*) containing measurements of the attributes. A more recently generated update contains a more current measurement. These updates are communicated over the network to a remote *monitor* that processes them for analytics and/or to provide any actuation that may be required. Majority of the IoT applications are cloud-backed, with the processing of measurements taking place in a remote cloud datacenter. Many such IoT subsystems can co-exist in the same physical space and use a shared and contended

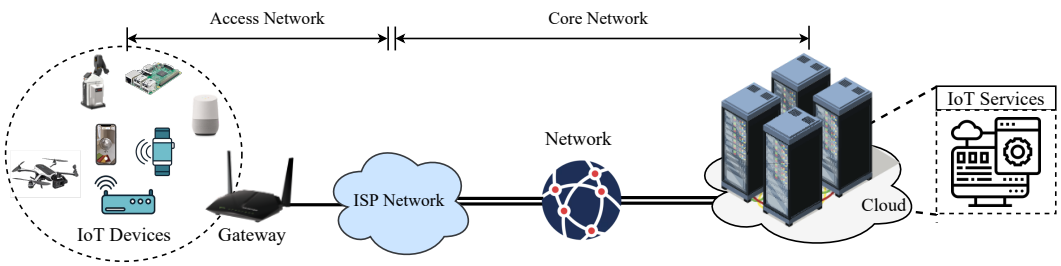


Figure 3.1: End-to-end network topology of cloud-based IoT services. IoT devices connect over a shared wireless network (Access Network) that connects to the Internet backbone via a gateway. These devices communicate updates via the Core Network to applications and services that operate in cloud datacenter infrastructures.

network which is managed by the serving ISP in the region (access network shown in Figure 3.1). The rest of the connection to the cloud is the backhaul core network and is widely known to be highly managed and reliable with significant bandwidth capacity [65].

For monitoring applications, it is desirable that *freshly* sensed information is available at monitors. However, as has been observed [2, 66, 67, 68], simply generating and sending updates at a high rate over the network is detrimental to this goal. Freshness at a monitor is optimized by the source smartly choosing an update rate as a function of the end-to-end network conditions. Freshness at the monitor suffers when a too small or a too large rate of updates is chosen by the source. Monitoring applications may achieve a low update packet delay by simply choosing a low rate at which the source sends updates. This, however, may be detrimental to freshness, as a low rate of updates can lead to a large *age* of sensed information at the monitor, simply because updates from the source are infrequent. On the other hand, a large rate of updates would have the monitor receive a steady stream of updates at a high rate. However, each update would have a high age as a result of it having experienced a large network delay.

The requirement of freshness is not akin to requirements of other pervasive real-time applications like voice and video. For these applications, the rate at which packets are sent is determined by the codec being used. Often, such applications adapt to network conditions by choosing an appropriate code rate. These applications, while resilient to packet drops to a certain degree, require end-to-end packet delays to lie within known limits and would like small end-to-end jitter. For instance, Real-time Transport Protocol (RTP) which is commonly used for delivering audio and video over IP, uses rate control mechanisms such as SCReAM [69] and GCC [70] to adapt the sending rate at available link capacity with minimum possible delay. More so than voice/video, monitoring applications are exceptionally loss resilient and they don't benefit from the source retransmitting lost updates. Instead, the source should continue sending new updates at its configured rate.

At the other end of the spectrum are applications like that of file transfer that require reliable transport and high throughputs but are delay tolerant. Such applications use the Transmission Control Protocol (TCP) for end-to-end delivery

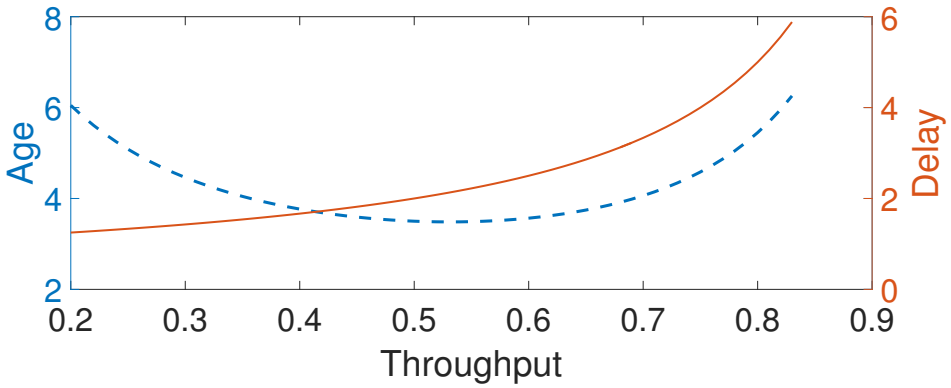


Figure 3.2: Interplay of the networking metrics of delay (solid line), throughput (normalized by service rate) and age. Shown for a M/M/1 queue [1] with service rate of 1. The age curve was generated using the analysis for a M/M/1 queue in [2].

of application packets. As we show in Section 3.3, the congestion control algorithm of TCP, which optimizes the use of the network pipe for throughput, and TCP's emphasis on guaranteed and ordered delivery is detrimental to keeping age low. Unlike TCP, User Datagram Protocol (UDP) ignores dropped packets and delivers packets to applications as soon as they are received. While this makes it desirable for age-sensitive applications; sending updates at a fixed rate incognizant of the underlying network can be, in fact, disastrous for age of the updates.

Figure 3.2 broadly captures the behavior of the metrics of delay and age as a function of throughput. Under light and moderate loads when packet dropping is negligible, throughput (average network utilization) increases linearly in the rate of updates. This leads to an increase in the average packet delay. Large packet delays coincide with large average age. Large age is also seen for small throughputs (and corresponding small rate of updates). At a low update rate, the monitor receives updates infrequently, and this increases the average age (staleness) of its most fresh update. Finally, observe that there exists a sending rate (and corresponding throughput) at which age is minimized.

Considering the growing need for applications to support fresh delivery of updates over the network and the inability of existing transport solutions to support the freshness requirement, we propose a novel transport layer protocol, namely the Age Control Protocol (ACP), which in a network-transparent manner regulates the rate at which updates from a source are sent over its end-to-end connection to the monitor. The goal is to keep the average age of sensed information at the

monitor to a minimum, where the age of an update is the time elapsed since its generation by the source. Based on feedback from the monitor, ACP adapts rate to the perceived congestion in the Internet. Consequently, ACP also limits congestion that would otherwise be introduced by sources sending to their monitors at unnecessarily fast update rates.

Our specific contributions include the following.

1. We demonstrate the inability of TCP to transport age-sensitive update packets over the Internet. We investigate the impact of different TCP configurations, such as congestion window and segment sizes on the age of updates at a monitor.

2. We detail the Age Control Protocol and how it interfaces with the TCP/IP networking stack.

3. We define the age control problem over the Internet. We intuit a good age control behavior using a mix of analysis and simulations. This leads us to a detailed description of the control algorithm of ACP. We also describe ACP+, which modifies ACP to achieve significantly better age control over a shared wireless access with high contention, which results from multiple end-to-end paths between sources and monitors sharing the same access.

4. We provide a detailed evaluation of ACP using a mix of simulations (controlled, easier to introduce very high contention, however, only a few hops) and real-world experiments over the Internet (WiFi access with many end-to-end paths sharing it, resulting in low to moderately high contention, followed by many hops over the very fast Internet backhaul).

5. We shed light on age control over end-to-end paths in the current Internet. We observe that the age optimizing rate over an end-to-end path that has a source send updates over a shared WiFi access followed by the Internet backhaul to a monitor in the cloud is much smaller than the bottleneck link rate of the path, which is the link rate of the WiFi access. The age optimizing rate stays at about 0.5 Mbps for WiFi access rates of 6 - 24 Mbps and backhaul rates as high as 200 Mbps. In fact, it is the age optimizing rate

over the path in the absence of a first WiFi hop. Turns out that the inter-continental path, much faster than the WiFi link, is in fact the constraining factor with respect to the achievable age over the end-to-end path. We also observe that at the age optimal rate, depending on the network scenario, a source may send multiple updates per round-trip-time (RTT) or may send an update over many RTT. In general, the bottleneck link rate and the baseline (updates sent in a stop-and-wait manner) RTT may not shed light on the age optimal rate. This in contrast to, the maximum end-to-end throughput, which the transport control protocol (TCP) would like to achieve, given by the product of the two quantities.

3.2 Related Work

The desire and need for timely updates arises in many fields, including, for example, vehicular updating [71], real-time databases [72], data warehousing [73], and web caching [74, 75]. In fact, the status update systems are very different than the data communication systems. While all packets are equally important in the latter, a packet is useful in the status update system only if it carries fresh information. The age of information (AoI) as a metric for *freshness* or *timeliness* has received a lot of attention in the past few years. While we try to discuss a few important works here, we also direct the reader to the recent surveys for more details on the literature [76, 77].

For sources sending updates to monitors, the AoI metric was first analyzed for elementary queues in [2]. To evaluate AoI for a single source sending updates through a network cloud [66] or through an M/M/k server [67, 68], out-of-order packet delivery was the key analytical challenge. A related (and generally more tractable) metric, peak age of information (PAoI), was introduced in [78] and properties of PAoI have also been studied in [79] for an FCFS M/G/1 multiclass queue and various M/M/1 queues that support preemption of updates in service or discarding of updates that find the server busy [78, 80]. Packet deadlines are found to improve AoI in [81]. AoI in the presence of errors is evaluated in [82] and for memoryless arrivals to a two-state Markov-modulated service process in [83]. Distributional properties of the age process have also been analyzed for

the D/G/1 queue under first-come-first-served (FCFS) [84], as well as single server FCFS and LCFS queues [85]. [86, 87] analyzes the general queueing systems of the form G/G/1/1. There have also been studies of energy-constrained updating [88, 89, 90, 91, 92, 93, 94, 95, 96].

There have also been substantial efforts to evaluate and optimize age for multiple sources sharing a communication link [97, 98, 99, 100, 101, 102, 103]. In particular, near-optimal scheduling based on the Whittle index has been explored in [97, 104, 105]. When multiple sources employ wireless networks subject to interference constraints, AoI has been analyzed under a variety of link scheduling methods [106, 107]. Analyzing the AoI for parallel server systems where updates are sent from a single source to a single-hop parallel server network was considered in [66, 68, 108]. The scheduling of updates for parallel server systems was considered in [109, 110, 111, 112, 113]. AoI analysis for multiple hop and multiple source networks has also received attention in [110, 114]. Notably, when updates arrive out of order, optimality properties of a Last Generated First Served (LGFS) service are found in [115]. Scheduling of a finite number of update packets in the presence of wireless interference for age optimization is an NP-hard problem [116]. [117] discusses the scheduling policies for minimizing peak and average age of information in wireless networks with time-varying links and under general interference constraints. [118, 107] take the effect of channel state information into account and propose an age-based scheduling policy.

There are works that design a sampling policy as a method to reduce the AoI [89, 90]. One such approach is the *zero-wait* policy that aims to achieve maximum throughput and minimum delay, but it fails to minimize the AoI, especially when the transmission times are heavy tail distributed [89, 90]. The optimal sampling policy in such cases is a threshold one, either deterministic or randomized. [119, 120] discusses the optimal threshold for real-time networks where both the forward and backward paths have random delays. Sampling policies for unreliable transmissions are considered in [121, 122]. More recently, [123] proposes an optimal sampling strategy to optimize data freshness for unreliable transmissions with random forward and backward channels. The proposed policy is based on a randomized threshold strategy where the source waits until the expected estimation error exceeds a threshold before sending a new sample in case of successful trans-

mission. Otherwise, the source sends a new update immediately without waiting. All these policies are some variation of stop and wait. Our work highlights the need to model multi-hop settings, wherein multiple updates could be queued at any time, to understand optimizing age over modern wide-area IP networks.

While the early work [71] explored practical issues such as contention window sizes, the subsequent AoI literature has primarily been focused on analytically tractable simple models. Moreover, a model for the system is typically assumed to be known. Our objective has been to develop end-to-end updating schemes that perform reasonably well without assuming a particular network configuration or model. This approach attempts to learn (and adapt to time variations in) the condition of the network links from source to monitor. This is similar in spirit to hybrid ARQ-based updating schemes [124, 125, 126] that learn the wireless channel. [125] uses the reinforcement techniques in unknown network environments. The chief difference is that hybrid ARQ occurs on the short timescale of a single update delivery while ACP learns what the network supports over many delivered updates.

Age in Systems. [127, 128] presents the first emulation study of age in wireless links. There is limited systems research on ageing of information and its optimization in real-world networks [129, 27, 130, 28, 131, 29, 132, 133]. In [129], authors discuss the age of information (AoI) in real-networks where a source is sending updates to a monitor over a selection of access networks including WiFi, LTE, 2G/3G and Ethernet. The authors observe a "U-shaped" AoI vs. arrival rate curve similar to the theoretical results for various non-preemptive queueing systems. The key takeaway from that work is the need for an AoI optimizer that can adapt to changing network topologies and delays. Our work, which we detail in the chapter, proposes the Age Control Protocol (ACP) [27, 28, 131], which is a transport-layer solution that works in an application-independent and network-transparent manner. ACP attempts to minimize the age of information of a source at a monitor connected via an end-to-end path over the Internet. In [29], we propose a modification to ACP and also compare it with other state-of-the-art TCP congestion control algorithms used in the Internet. In [132], WiFresh, a MAC and application-layer solution to ageing of updates over a wireless network is proposed. While both [28] and [132] look at ageing of updates on the Internet, they differ

in their approach and scope. ACP is a transport layer solution that works by adapting the source generation rate without any specific knowledge of the access network or any network hop to the monitor, whereas, WiFresh is a scheduling solution designed for WiFi networks.

In [130, 134], the authors discuss the issues and challenges associated with measuring AoI in real networks, including synchronization, selection of hardware, and choice of transport protocol and draw insights for AoI aware transmission protocols. A detailed analysis of all the age related practice works on real networks can be found in [135].

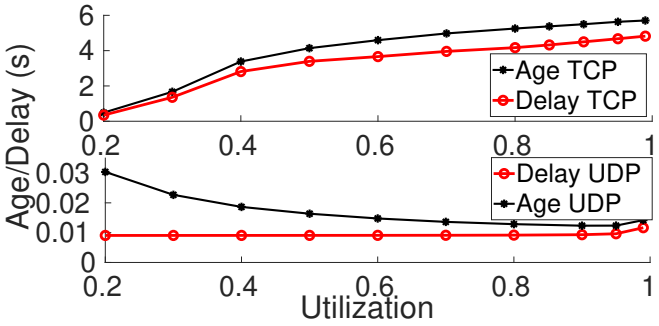
In summary, unlike other works, ACP aims to provide a practical age control algorithm that aims to minimize the average age in a multi-source and multi-hop network where there is no prior information about the network and no control over the other sources that have access to it.

3.3 Age Sensitive Update Traffic over TCP

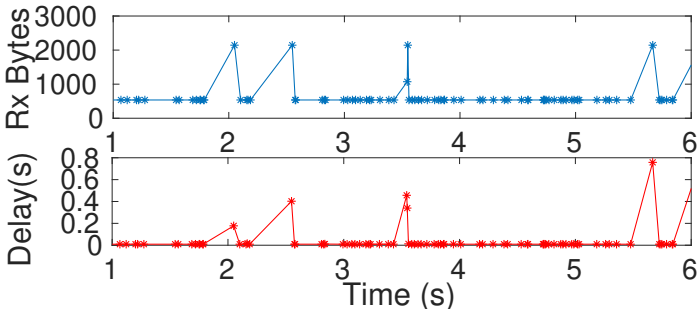
Before we delve into the problem of end-to-end age control, we demonstrate why TCP as a choice of transport protocol is unsuitable for age-sensitive traffic. Specifically, we show that the congestion control mechanism of TCP, together with its goal of guaranteed and ordered delivery of packets, can lead to a very high age at the monitor, in comparison to when UDP is used, for a wide range of utilization of the network by the traffic generated by the source, and not just when the utilization is high.

We simulated a simple network consisting of a source that sends measurement updates to a monitor via a single Internet Protocol (IP) router. The source node has a bidirectional point-to-point (P2P) link of rate 1 Mbps to the router. A similar link connects the router to the monitor. The source uses a TCP client to connect to a TCP server at the monitor and sends its update packets over the resulting TCP connection. We will also compare the obtained age with when UDP is used instead.

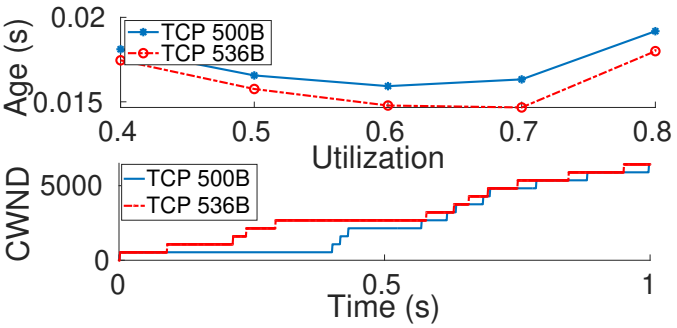
Retransmissions and In-order Delivery: Figure 3.3a illustrates the impact of



(a) Impact of packet errors



(b) Timestamp of TCP receiver delays



(c) Impact of packet size

Figure 3.3: Impact of packet errors, receiver delays and packet size on age when using TCP and UDP.

packet error on TCP. A packet was dropped independently of other packets with a probability of 0.1. The figure compares the average age at the monitor and the average update packet delay, which is the time elapsed between the generation of a packet at the source and its delivery at the monitor when using TCP and UDP. On using TCP, the time average age achieves a minimum value of 0.18 seconds when the source utilizes a fraction 0.2 of the available 1 Mbps to send update packets. This is clearly much larger than the minimum age of ≈ 0.01 seconds at the utilization of ≈ 0.8 when UDP is used.

The large minimum age when using TCP is explained by the way TCP guar-

antees in-order packet delivery to the receiving application (monitor). It causes fresher updates that have arrived out-of-order at the TCP receiver to wait for older updates that have not yet been received, for example, because of packet losses in the network. This can be seen in Figure 3.3b that shows how large measured packet delays coincide with a spike in the number of bytes received by the monitor application. The large delay is that of a received packet that had to undergo a TCP retransmission. The corresponding spike in received bytes, which is preceded by a pause, is because bytes with fresher information received earlier but out of order are held by the TCP receiver till the older packet is received post retransmission. Unlike TCP, UDP ignores dropped packets and delivers packets to applications as soon as they are received. This makes it desirable for age-sensitive applications. As we will see later, ACP uses UDP to provide update packets with end-to-end transport over IP.

TCP Congestion Control and Small Packets: Next, we describe the impact of small packets on the TCP congestion algorithm and its impact on age. This is especially relevant to a source sending measurement updates as the resulting packets may have small application payloads. Note that no packet errors were introduced in simulations used to make the following observations. Observe in the upper plot of Figure 3.3c that the 500 byte packet payloads experience higher age at the monitor than the larger 536 byte packets. The reason is explained by the impact of packet size on how quickly the size of the TCP congestion window (CWND) increases. The congestion window size doesn't increase till a sender maximum segment size (SMSS) bytes are acknowledged. TCP does this to optimize the overheads associated with sending payload. Packets with fewer bytes may thus require multiple TCP `ACK`(s) to be received for the congestion window to increase. This explains the slower increase in the size of the congestion window for 500 byte payloads seen in Figure 3.3c. This causes smaller packets to wait longer in the TCP send buffer before they are sent out by the TCP sender, which explains the larger age in Figure 3.3c.

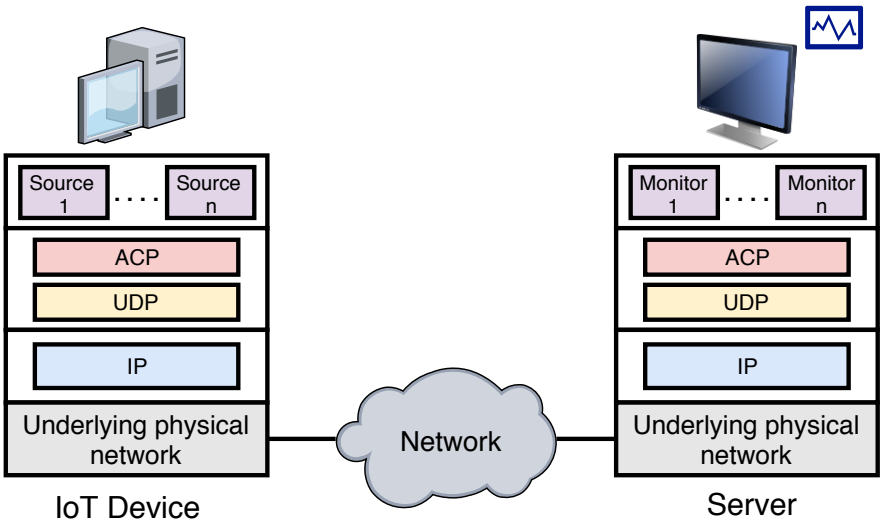


Figure 3.4: The ACP end-to-end connection.

3.4 The Age Control Protocol

The Age Control Protocol resides in the transport layer of the TCP/IP networking stack and operates only on the end hosts. Figure 3.4 shows an *end-to-end connection* between two hosts, an IoT device and a server over the Internet. A source opens an ACP connection to its monitor. Multiple sources may connect to the same monitor. ACP uses the unreliable transport provided by the user datagram protocol (UDP) for sending updates generated by the sources. This is in line with the requirements of fresh delivery of updates. Retransmissions make an update stale and also compete with fresh updates for network resources.

The source ACP appends a header to an update from a source. The header contains a *timestamp* field that stores the time the update was generated. The source ACP suggests to the source the rate at which it must generate updates. To be able to calculate the rate, the source ACP must estimate network conditions over the end-to-end path to the monitor ACP. This is achieved by having the monitor ACP acknowledge each update packet received from the source ACP by sending an `ACK` packet in return. The `ACK` contains the timestamp of the update being acknowledged. The `ACK`s allow the source ACP to keep an estimate of the age of sensed information at the monitor. An *out-of-sequence* `ACK`, which is an `ACK` received after an `ACK` corresponding to a more recent update packet, is discarded by the source ACP. Similarly, an update that is received *out-of-sequence* is discarded by the monitor. This is because the monitor has already received a more recent

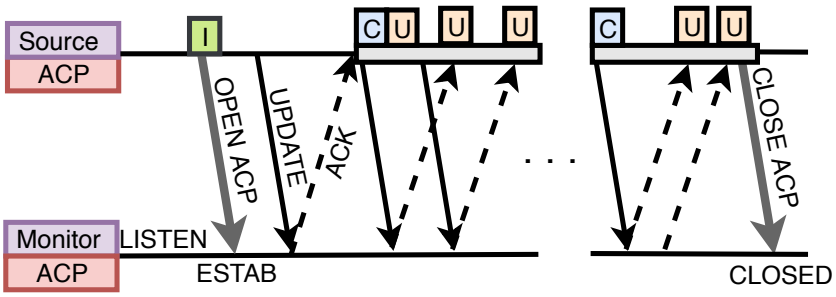


Figure 3.5: Timeline of an ACP connection. I marks the beginning of the initialization phase. C denotes the control algorithm (Algorithm 2 or 3) executed when a new control epoch begins. U is executed when an `ACK` is received and updates $\overline{Z}, \overline{\text{RTT}}$, and $\mathcal{T}$.

measurement from the source.

Figure 3.5 shows a timeline of a typical ACP connection. For an ACP connection to take place, the monitor ACP must be listening on a previously advertised UDP port. The ACP source first establishes a UDP connection with the monitor. This is followed by an *initialization* phase during which the source sends an update and waits for an `ACK` or for a suitable timeout to occur, and repeats this process for a few times, with the goal of probing the network to set an initial update rate. Following this phase, the ACP connection may be described by a sequence of *control epochs*. The end of the *initialization* phase marks the start of the first control epoch. At the beginning of each control epoch, ACP sets the rate at which updates generated from the source are sent until the beginning of the next epoch.

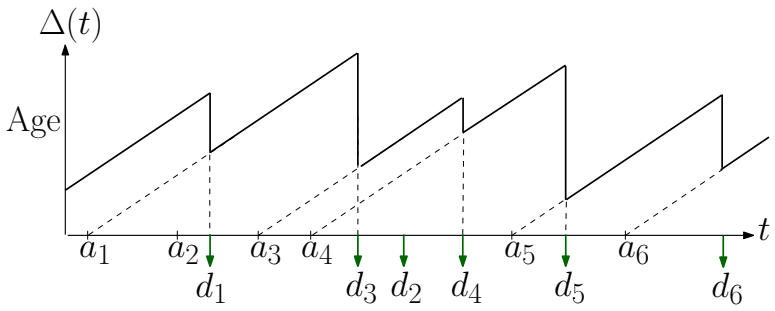


Figure 3.6: A sample function of the age $\Delta(t)$. Updates are indexed $1, 2, \ldots$. The timestamp of update i is a_i. The time at which update i is received by the monitor is d_i. Since update 2 is received out-of-sequence, it doesn't reset the age process.

3.5 The Age Control Problem

We will formally define the age of sensed information at a monitor. To simplify the presentation, in this section, we will assume that the source and monitor are time synchronized, although the functioning of ACP doesn't require the same. Let $z(t)$ be the timestamp of the freshest update received by the monitor up to time t. Recall that this is the time the update was generated by the source.

The age at the monitor is $\Delta(t) = t - z(t)$ of the freshest update available at the monitor at time t. An example sample function of the age stochastic process is shown in Figure 3.6. The figure shows the timestamps $a_1, a_2, \ldots, a_6$ of 6 packets generated by the source. Packet i is received by the monitor at time d_i. At time d_i, packet i has age $d_i - a_i$. The age $\Delta(t)$ at the monitor increases linearly in between reception of updates received in the correct sequence. Specifically, it is reset to the age $d_i - a_i$ of packet i, in case packet i is the freshest packet (one with the most recent timestamp) at the monitor at time d_i. For example, when update 3 is received at the monitor, the only other update received by the monitor until then was update 1. Since update 1 was generated at time $a_1 < a_3$, the reception of 3 resets the age to $d_3 - a_3$ at time d_3. On the other hand, while update 2 was sent at a time $a_2 < a_3$, it is delivered out-of-order at a time $d_2 > d_3$. So packet 2 is discarded by the monitor ACP and age stays unchanged at time d_2.

We want to choose the rate λ (updates/second) that minimizes the expected value $\lim_{t\to\infty} E[\Delta(t)]$ of age at the monitor, where the expectation is over any randomness introduced by the network. Note that in the absence of a priori knowledge of a network model, as is the case with the end-to-end connection over which ACP runs, this expectation is unknown to both source and monitor and must be estimated using measurements. Lastly, we would like to dynamically adapt the rate λ to nonstationarities in the network.

3.6 Good Age Control Behavior and Challenges

ACP must suggest a rate λ updates/second at which a source must send fresh updates to its monitor. ACP must adapt this rate to network conditions. To

build intuition, let's suppose that the end-to-end connection is well described by an idealized setting that consists of a single FCFS queue that serves each update in constant time. An update generated by the source enters the queue, waits for previously queued updates, and then enters service. The monitor receives an update once it completes service. Note that every update must age at least by the (constant) time it spends in service, before it is received by the monitor. It may age more if it ends up waiting for one or more other updates to complete service.

In this idealized setting, one would want a new update to arrive as soon as the last generated update finishes service. To ensure that the age of each update received at the monitor is the minimum, one must choose a rate λ such that new updates are generated in a periodic manner with the period set to the time an update spends in service. Also, update generation must be synchronized with service completion instants so that a new update enters the queue as soon as the last update finishes service. In fact, such a rate λ is age minimizing even when updates pass through a sequence of $Q > 1$ such queues in tandem [1]. The update is received by the monitor when it leaves the last queue in the sequence. The rate λ will ensure that a generated packet ages exactly Q times the time it spends in the server of any given queue. At any given time, there will be exactly Q update packets in the network, one in each server.

Of course, the assumed network is a gross idealization. We assumed a series of similar constant service time facilities and that the time spent in service and instant of service completion were known exactly. We also assumed a lack of any other traffic. However, as we will see further, the resulting intuition is significant. Specifically, *a good age control algorithm must strive to have as many update packets in transit as possible while simultaneously ensuring that these updates avoid waiting for other previously queued updates.*

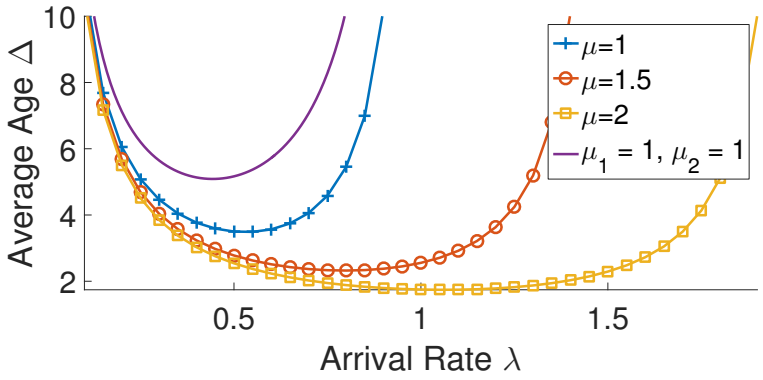
Before we detail our proposed control method, we will make a few salient observations using analytical results for simple queueing models and simulation results that capture stochastic service and generation of updates. These will help build on our intuition and also elucidate the challenges of age control over a priori unknown and likely non-stationary end-to-end network conditions.

3.6.1 Analytical Queueing Model for Two Queues

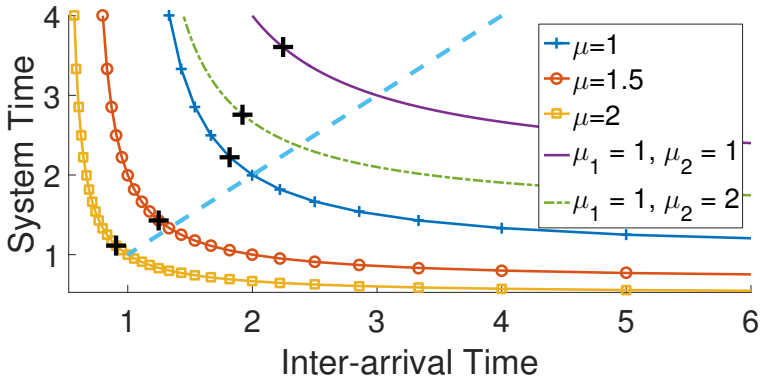
We will consider two queueing models. One is the $M/M/1$ FCFS queue with an infinite buffer in which a source sends update packets at a rate λ to a monitor via a single queue, which services packets at a rate μ updates per second. The updates are generated as a Poisson process of rate λ and packet service times are exponentially distributed with $1/\mu$ as the average time it takes to service a packet. In the other model, updates travel through two queues in tandem. Specifically, they enter the first queue that is serviced at the rate μ_1. On finishing service in the first queue, they enter the second queue that services packets at a rate of μ_2. As before, updates arrive to the first queue as a Poisson process and packet service times are exponentially distributed. The average age for the case of a single $M/M/1$ queue was analyzed in [2]. We extend their analysis to obtain analytical expressions of average age as a function of λ, μ_1 and μ_2 for the two queue case, by using the well-known result that updates also enter the second queue as a Poisson process of rate λ [1]. Note that packets arrive into queue 2 as a Poisson process of rate λ. We assumed that a packet that arrives into queue 2 undergoes service for a time that is independent of the time it spent in service in server 1. That is, somehow, correlations that may be introduced due to packet lengths do not exist and both the queues can be analyzed as $M/M/1/\infty$ queues. The derivation is in [30, Appendix A].

On the impact of non-stationarity and transient network conditions: Figure 3.7a shows the expected value (average) of age as a function of λ when the queueing systems are in *steady state*. It is shown for three single $M/M/1$ queues, each with a different service rate, and for two queues in tandem with both servers having the same unit service rate. Observe that all the age curves have a bowl-like shape that captures the fact that a too small or a too large λ leads to large age. Such behavior has been observed in non-preemptive queueing disciplines in which updates can't preempt other older updates. A reasonable strategy to find the optimal rate thus seems to be one that starts at a certain initial λ and changes λ in a direction such that a smaller expected age is achieved.

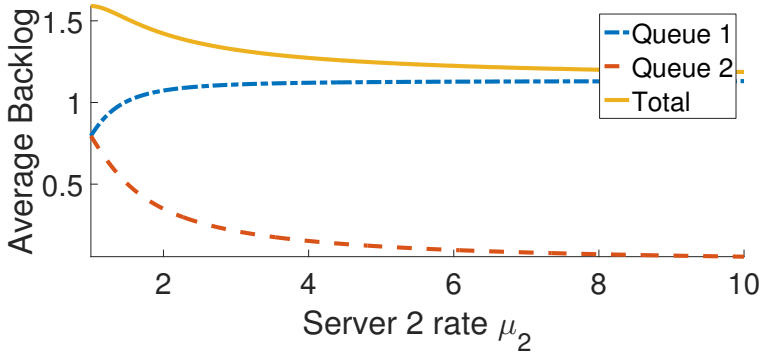
In practice, the absence of a network model (unknown service distributions and expectations), would require Monte-Carlo estimates of the expected value of age for



(a) Average Age as a function of update rate λ



(b) Average system time as a function of inter-arrival time $(1/\lambda)$



(c) Average backlog as μ_2 increases from 1 to 5. $\mu_1 = 1$

Figure 3.7: Analytical queueing model analysis for two queues having service rates μ_1 and μ_2.

every choice of λ. Getting these estimates, however, would require averaging over a large number of instantaneous age samples and would slow down adaptation. This could lead to updates experiencing excessive waiting times when λ is too large. Worse, transient network conditions (a run of bad luck) and non-stationarities, for example, because of introduction of other traffic flows, could push these delays to even higher values, leading to an even larger backlog of packets in transit. Figure 3.7a, illustrates how changes in network conditions (service rate μ and number of hops (queues)) can lead to large changes in the expected age.

It is desirable for a good age control algorithm to not allow the end-to-end connection to drift into a high backlog state. As we describe in the next section, ACP tracks changes in the average number of backlogged packets and average age over short intervals, and in case backlog and age increase, ACP acts to rapidly reduce the backlog.

On Optimal Average Backlogs: Figure 3.7b plots the average packet system times, where the system time of a packet is the time that elapses between its arrival and completion of its service, as a function of inter-arrival time ($1/\lambda$) for three single queue $M/M/1$ networks and two networks that have two queues in tandem. As expected, an increase in inter-arrival time reduces the system time. As inter-arrival times become large, packets wait less often for others to complete service. As a result, as inter-arrival time becomes large, the system times converge to the average service time of a packet.

For each queueing system, Figure 3.7b also marks with (+) the average inter-arrival time $1/\lambda^*$ that minimizes age. It is instructive to note that for the three single queue systems this inter-arrival time is only slightly smaller than the system time (The blue dashed line in Figure 3.7b is at 45°). However, for the two queues in tandem with service rates of 1 each, the inter-arrival time is much smaller than the system time. The implication being that on an average it is optimal to send slightly more than one (≈ 1.12) packet every system time for the single queue system. However, for the two queue network with the similar servers, we want to send a larger number (≈ 1.6) of packets every system time. For the two queue network where the second queue is served by a faster server, this number is smaller (≈ 1.43). As we observe next, as one of the servers becomes faster, the two queue network becomes more akin to a single queue network with the slower server.

Figure 3.7c shows how the optimal average backlog varies as a function of service rate(s). For each choice of service rate (i.e., μ in the single queue or μ_1, μ_2 in the tandem queue), the age-optimal arrival rate λ^* is selected. For fixed $\mu_1 = 1$, λ^* increases as μ_2 increases, and this causes the average backlog in queue 1 to increase. However, as λ^* increases more slowly than μ_2, the backlog in queue 2 decreases, while the sum backlog approaches the optimal backlog for the single queue system. Specifically, as queue 1 becomes a larger rate bottleneck relative to

Network	R_1	R_2	R_3	R_4	R_5	R_6
Net A	1	1	1	1	1	1
Net B	1	1	5	5	1	1
Net C	1	5	5	5	5	1
Net D	5	5	5	5	5	1
Net E	5	5	5	5	5	5

Table 3.1: Various P2P link configurations applied to the network diagram in Figure 3.12. The rates R_i are in Mbps. R_1 is the rate of the link between the source and AP-1 and R_6 is that of the link between AP-2 and the monitor.

queue 2, the optimal λ^* must adapt to the bottleneck queue.

These observations stay the same on swapping μ_1 and μ_2. Moreover, when the rates μ_1 and μ_2 are similar, the queues see similar backlogs. Notably, when $\mu_1 = \mu_2 = 1$, the backlog per queue is smaller than in a network with only a single such queue. However, the sum backlog (≈ 1.6) is larger.

3.6.2 Simulating Larger Number of Hops

To see if this intuition generalizes to more number of hops, we simulated an end-to-end connection which has the source send its packet to the monitor over 6 hops, where each hop is serviced by a bidirectional P2P link. The hops are shown in Figure 3.12. We vary the rates at which the P2P links transmit packets to gain insight into how queues in a network must be populated with update packets at an age optimal rate. We also introduce other traffic in the network that occupies, on an average, a fraction 0.2 Mbps of each P2P link from the source to the monitor. The different configurations are summarized in Table 3.1. For each network configuration, we have the source send updates over UDP to the monitor using an a priori chosen rate λ. We vary λ over a wide range of values and for each λ, we calculate the obtained time average age. These simulations allow us to empirically pick the age minimizing λ for the given network.

Figure 3.8 shows the time average *backlog* (queue occupancy) at the different nodes in the network at the optimal λ. The backlog at a node includes the update packet being transmitted on a node's outgoing P2P link and any update that is

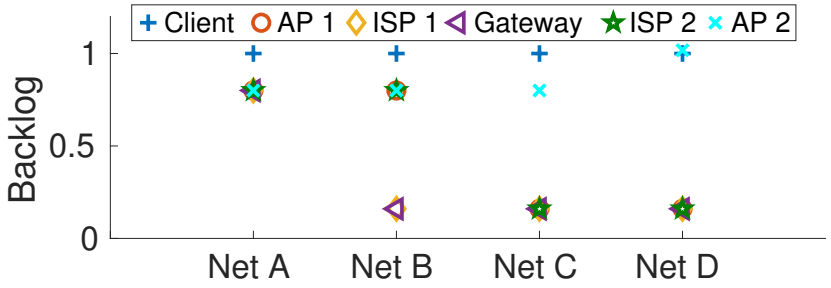


Figure 3.8: Average backlogs at different nodes in the network, shown in Fig 3.12, at the optimal update rate. Net E is similar to Net A and not shown.

awaiting transmission at the node. Observe that all P2P links in each of Net A and Net E have the same rate, 1 and 5 Mbps, respectively. Though Net B has links much faster than that of Net A, for both these networks the average backlog at all nodes is close to 1. That it is smaller than 1 is explained by the presence of the other flow. The other flow, which also originates at the client, is also the reason why the client sees a slightly larger average queue occupancy by the update packets.

Net B has faster P2P links connecting ISP(s) and the Gateway when compared to Net A. However, its other links are slower than that in Net E. We see that the nodes that have fast outgoing links have low backlogs and those that have slow links have an average backlog close to 0.8. The source has a slow outgoing link and as a result of the other flow sees slightly larger occupancy of update packets. In summary, at λ that minimizes average age, as is also shown in Figure 3.8 for Net C and Net D, *nodes with outgoing links that are bottlenecks relative to the others' links see a backlog such that no more than one update packet is queued at them.* Naturally, nodes with faster links see smaller backlogs in proportion to how fast their links are with respect to the bottleneck.

A corollary to the above observations is that a good age control algorithm should on an average have a larger number of packets simultaneously in transit in a network with a larger number of hops (nodes/queues).

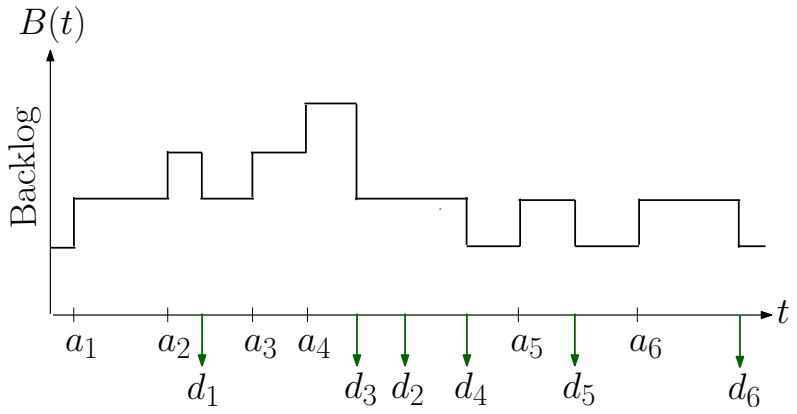


Figure 3.9: A sample function of the backlog process $B(t)$. Updates are indexed $1, 2, \ldots$. The timestamp of update i is a_i. The time at which update i is received by the monitor is d_i. Since update 3 is received before 2, backlog is reduced by 2 packets at d_3. Also, there is no change in $B(t)$ at $d_2 > d_3$.

3.7 The ACP Control Algorithm

Let the control epochs of ACP (Section 3.5) be indexed $1, 2, \ldots$. Epoch k starts at time t_k. At t_1 the update rate λ_1 is set to the inverse of the average packet round-trip-times (RTT) obtained at the end of the initialization phase. At time t_k, $k > 1$, the update rate is set to λ_k. The source transmits updates at a fixed period of $1/\lambda_k$ in the interval (t_k, t_{k+1}).

Let $\overline{\Delta}_k$ be the estimate at the source ACP of the time average update age at the monitor at time t_k. This average is calculated over (t_{k-1}, t_k). To calculate it, the source ACP must construct its estimate of the age sample function (see Figure 3.6), over the interval, at the monitor. It knows the time a_i the source sent a certain update i. However, it needs the time d_i at which update i was received by the monitor, which it approximates by the time the `ACK` for packet i was received. On receiving the `ACK`, it resets its estimate of age to the resulting round-trip-time (RTT) of packet i. Note that this value is an overestimate of the age of the update packet when it was received at the monitor, since it includes the time taken to send the `ACK` over the network. The time average $\overline{\Delta}_k$ is obtained simply by calculating the area under the resulting age curve over (t_{k-1}, t_k) and dividing it by the length $t_k - t_{k-1}$ of the interval.

Let $\overline{B}_k$ be the time average of backlog calculated over the interval (t_{k-1}, t_k).

This is the time average of the instantaneous backlog $B(t)$ over the interval. The instantaneous backlog increases by 1 when the source sends a new update. When an `ACK` corresponding to an update i is received, update i and any unacknowledged updates older than i are removed from the instantaneous backlog. Figure 3.9 shows the instantaneous backlog as a function of time corresponding to the age sample function in Figure 3.6.

In addition to using RTT(s) of updates for age estimation, we also use them to maintain an exponentially weighted moving average (EWMA) $\overline{\text{RTT}}$ of RTT. We update $\overline{\text{RTT}} = (1-\alpha)\overline{\text{RTT}} + \alpha\text{RTT}$ on reception of an `ACK` that corresponds to a round-trip-time of RTT. The source ACP also estimates the inter-update arrival times at the monitor and the corresponding EWMA $\overline{Z}$. The inter-update arrival times are approximated by the corresponding inter-ACK arrival times. The length $\overline{T}$ of a control epoch is set as an integral multiple of $\mathcal{T} = \min(\overline{\text{RTT}}, \overline{Z})$. This ensures that the length of a control epoch is never too large and allows for fast enough adaptation. Note that at a sufficiently low rate λ_k of sending updates $\overline{Z}$ is large and at a sufficiently high update rate $\overline{\text{RTT}}$ is large. At time t_k we set $t_{k+1} = t_k + \overline{T}$. In all our evaluation we have used $\overline{T} = 10\mathcal{T}$. The resulting length of $\overline{T}$ was observed to be long enough to see desired changes in average backlog and age in response to a choice of source update rate at the beginning of an epoch. The source updates $\overline{\text{RTT}}$, $\overline{Z}$, and $\overline{T}$ every time an `ACK` is received.

At the beginning of control epoch $k > 1$, at time t_k, the source ACP calculates the difference $\delta_k = \overline{\Delta}_k - \overline{\Delta}_{k-1}$ in average age measured over intervals (t_{k-1}, t_k) and (t_{k-1}, t_{k-2}) respectively. Similarly, it calculates $b_k = \overline{B}_k - \overline{B}_{k-1}$.

ACP at the source chooses an action u_k at the k^{th} epoch that targets a change b^*_{k+1} in average backlog over an interval of length $\mathcal{T}$ with respect to the k^{th} interval. The actions, may be broadly classified into additive increase (INC), additive decrease (DEC), and multiplicative decrease (MDEC). MDEC corresponds to a set of actions MDEC(γ), where $\gamma = 1, 2, \ldots$. We have

$$\begin{aligned} &\text{INC: } b^*_{k+1} = \kappa, \text{ DEC: } b^*_{k+1} = -\kappa, \\ &\text{MDEC}(\gamma)\text{: } b^*_{k+1} = -(1-2^{-\gamma})B_k, \end{aligned} \tag{3.1}$$

where $\kappa > 0$ is a step size parameter.

Algorithm 2 Control Algorithm of ACP

INPUT: $b_k, \delta_k, \overline{T}, B_k$
INIT: $flag \leftarrow 0$, $\gamma \leftarrow 0$
while true **do**
  if $b_k > 0$ && $\delta_k > 0$ **then**
    if $flag == 1$ **then**
      $\gamma = \gamma + 1$
      MDEC(γ): $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
    else
      DEC: $b^*_{k+1} = -\kappa$
    end if
    $flag \leftarrow 1$
  else if $b_k > 0$ && $\delta_k < 0$ **then**
    if $flag == 1$ && $|b_k| < 0.5 * |b^*_k|$ **then**
      $\gamma = \gamma + 1$
      MDEC(γ): $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
    else
      INC: $b^*_{k+1} = \kappa$, $flag \leftarrow 0$, $\gamma \leftarrow 0$
    end if
  else if $b_k < 0$ && $\delta_k > 0$ **then**
    INC: $b^*_{k+1} = \kappa$, $flag \leftarrow 0$, $\gamma \leftarrow 0$
  else if $b_k < 0$ && $\delta_k < 0$ **then**
    if $flag == 1$ && $\gamma > 0$ **then**
      MDEC(γ): $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
    else
      DEC: $b^*_{k+1} = -\kappa$, $flag \leftarrow 0$, $\gamma \leftarrow 0$
    end if
  end if
  `UpdateLambda`(b^*_{k+1})
  wait $\overline{T}$
end while

function `UpdateLambda`(b^*_{k+1})
$\lambda_k = \frac{1}{\overline{Z}} + \frac{b^*_{k+1}}{\mathcal{T}}$
return λ_k

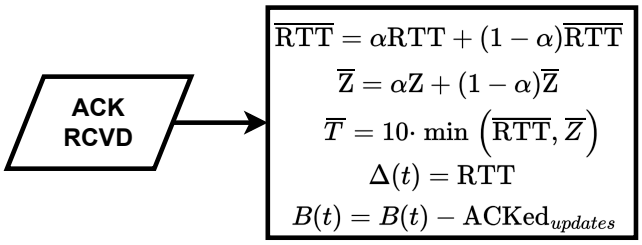


Figure 3.10: Update of $\overline{\mathrm{RTT}}$, $\overline{Z}$, $\overline{T}$, $\Delta(t)$, and $B(t)$, which takes place every time an `ACK` is received by the source ACP.

ACP attempts to achieve b^*_{k+1} by setting λ_k appropriately. The estimate of $\overline{Z}$ at the source ACP of the average inter-update arrival time at the monitor gives us the rate $1/\overline{Z}$ at which updates sent by the source arrive at the monitor. This and λ_k allow us to estimate the average change in backlog over $\mathcal{T}$ as $(\lambda_k - (1/\overline{Z}))\mathcal{T}$. Therefore, to achieve a change of b^*_{k+1} requires choosing $\lambda_k = \frac{1}{\overline{Z}} + \frac{b^*_{k+1}}{\mathcal{T}}$. Algorithm 2

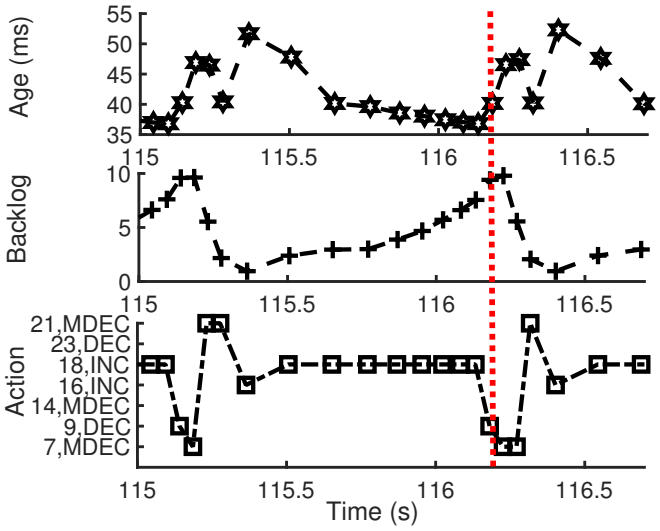


Figure 3.11: A snippet from the function of ACP. The y-axis of the plot showing actions denotes the action and the line number in Algorithm 2. Note the action marked by the dotted red line. At the time instant ACP observes an increase in both backlog and age and chooses (9,DEC) initially. However, there is still a significant jump in age. This results in the choice of multiplicative decrease (7,MDEC).

summarizes how ACP chooses its action u_k as a function of b_k and δ_k. Figure 3.10 summarizes updates on receipt of an `ACK`.

The source ACP targets a reduction in average backlog over the next control interval in case either $b_k > 0, \delta_k > 0$ or $b_k < 0, \delta_k < 0$. The first condition (line 4) indicates that the update rate is such that updates are experiencing larger than optimal delays. ACP attempts to reduce the backlog, first using DEC (line 9), followed by multiplicative reduction MDEC to reduce congestion delays and in the process reduce age quickly. Consecutive occurrences ($flag == 1$) of this case (tracked by increasing γ by 1 in line 6) attempt to decrease backlog even more aggressively, by a larger power of 2.

The condition $b_k < 0, \delta_k < 0$ occurs on a reduction in both age and backlog. ACP greedily aims at reducing backlog further hoping that age will reduce too. It attempts MDEC (line 23) if previously the condition $b_k > 0, \delta_k > 0$ was satisfied. Else, it attempts an additive decrease DEC.

The source ACP targets an increase in average backlog over the next control interval in case either $b_k > 0, \delta_k < 0$ or $b_k < 0, \delta_k > 0$. On the occurrence of the first condition (line 20) ACP greedily attempts to increase backlog.

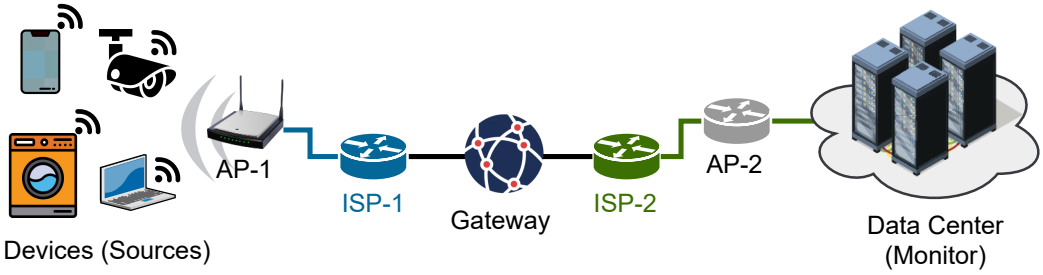


Figure 3.12: Sources are connected to the monitor via multiple routers and access points. Each source update travels over six hops. The first hop is between the source and access point AP-1. This could be either P2P or WiFi. The other hops that involve the ISP(s) and the Gateway are an abstraction of the Internet. These hops are P2P links and we vary their rates to simulate different end-to-end RTT.

When the condition $b_k < 0, \delta_k > 0$ occurs, we check if the previous action attempted to reduce the backlog. If not, it hints at too low an update rate causing an increase in age. So, ACP attempts an additive increase (line 17) of backlog. If yes, and if the actual change in backlog was much smaller than the desired (line 13), ACP attempts to reduce backlog multiplicatively. This helps counter situations where the increase in age is in fact because of increasing congestion. Specifically, increasing congestion in the network may cause the inter-update arrival rate $1/\overline{Z}$ at the monitor to reduce during the epoch. As a result, despite the attempted multiplicative decrease in backlog, it may change very little. Clearly, in such a situation, even if the backlog reduced a little, the increase in age was not caused because the backlog was low. The above check ensures ACP attempts reducing backlog to desired levels. In the above case, if instead ACP ignores the much smaller than desired change, it will end up increasing the rate of updates, further increasing backlog and age. Figure 3.11 shows a snippet of ACP in action.

3.8 Evaluation Methodology

We used a mix of real-world experiments and simulations to evaluate ACP. While the real-world experiments allowed us to test ACP over an intercontinental end-to-end connection, simulations allowed us to test with large numbers of sources contending with each other over a shared wireless access under varied wireless channel conditions and densities of source placements.

We evaluated ACP in the real-world by having $1-10$ sources connected to an

802.11g WiFi access point send their updates over the Internet to monitors that were running on an ec2 AWS Frankfurt [136] server with a global IP. The WiFi access point was a part of IIIT-Delhi's enterprise network that provides wireless access at the university campus. This setup allowed us to test ACP over a path with large RTT(s) and tens of hops. While the WiFi access point had only our test sources connected to it, we don't control the interference that may be created by adjoining access points or WiFi clients. Lastly, we had no control over the traffic on the university intranet when the experiments were performed.

Figure 3.12 shows the end-to-end network used for simulations. We start by describing the wireless access over which sources connect to AP-1. We performed simulations for $1 - 50$ sources accessing AP-1 using the WiFi (802.11g) medium access. We simulated for sources spread uniformly and randomly over areas of 10×10 m^2, 20×20 m^2 and 50×50 m^2. The channel between a source and AP-1 was chosen to be Log-Normally distributed with choices of 4, 8, and 12 for the standard deviation. The pathloss exponent was 3. WiFi physical (PHY) layer rates were set to one of 12 Mbps and 54 Mbps.

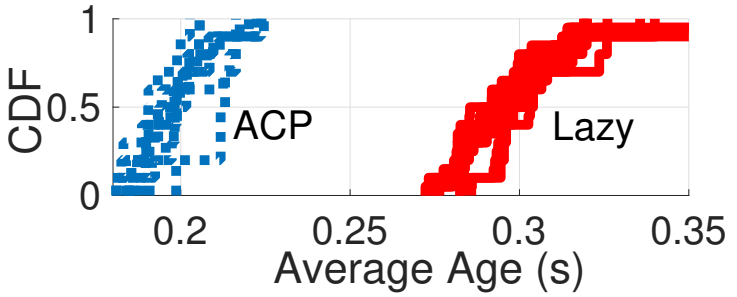
For the network beyond AP-1, all links were configured to be P2P. We set the P2P link rates from the set $\{0.3, 0.6, 1.2, 6.0\}$ Mbps. This was to simulate network RTT of a wide range. We used the network simulator ns3[1] together with the YansWiFiPhyHelper[2]. Our simulated network is, however, limited in the number of hops, which is six.

To compare the age control performance of ACP, we use *Lazy*. *Lazy*, like ACP, also adapts the update rate to network conditions. However, it is very conservative and keeps the average number of update packets in transit small. Specifically, it updates the $\overline{\text{RTT}}$ every time an `ACK` is received and sets the current update rate to the inverse of $\overline{\text{RTT}}$. Thus, it aims at maintaining an average backlog of 1.

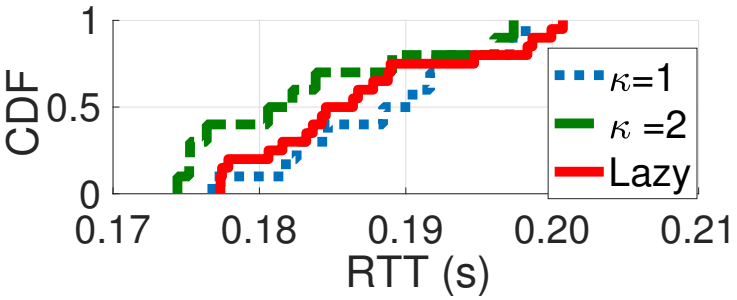
We end by stating that an appropriate selection of step size κ is crucial to the proper functioning of ACP. We chose it by trial and error. For simulations, we found a step size of $\kappa = 0.25$ to be the best. However, this turned out to be too small for experiments over the Internet. For these, we tried $\kappa \in \{1, 2\}$.

[1] https://www.nsnam.org/

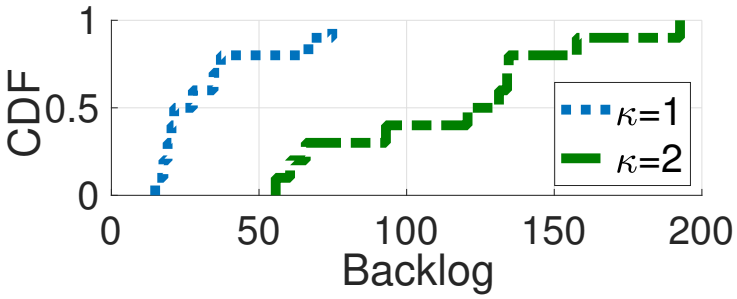
[2] https://www.nsnam.org/doxygen/classns3_1_1_yans_wifi_phy.html



(a) Average Age



(b) Average RTT



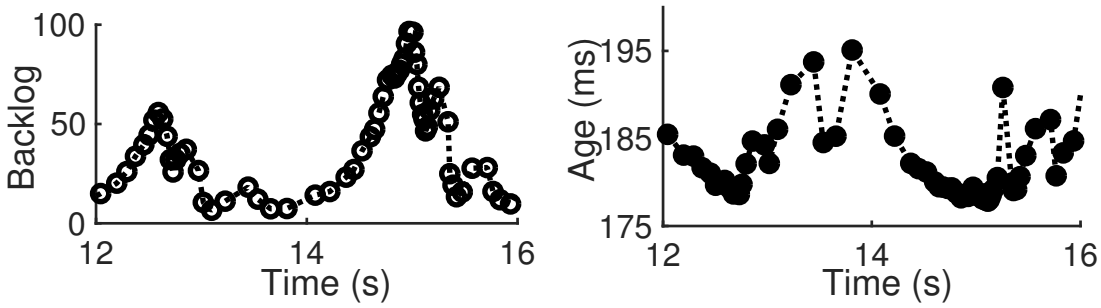
(c) Average Backlog

Figure 3.13: Comparison of *Lazy* and ACP with step size choices of $\kappa = 1, 2$ obtained over 10 runs each. The Age CDF(s) of all the 10 sources are shown.

Next, we will discuss the real-world results followed by the simulation results.

3.9 Inter-Continental Updates

In this section, we will show results for when 10 sources sent their updates to monitors on the configured AWS ec2 server. The sources, as described earlier, gained access to the Internet via an enterprise access point. The results were obtained by running ACP and *Lazy* alternately for 10 runs. Each run was restricted to 1000 update packets long so that on an average ACP and *Lazy* experienced similar network conditions. We ran ACP for $\kappa = 1$ and $\kappa = 2$. Using `traceroute` [137], we observed that the number of hops was large, about 30, during these experiments.



(a) Average Backlog (b) Average Age

Figure 3.14: The time evolution of average backlog and age that resulted from one of the ACP source sending updates over the Internet.

Figure 3.13 summarizes the comparison of ACP and *Lazy*. Figure 3.13a shows the cumulative distribution functions (CDF) of the average age obtained by each source when using ACP (using $\kappa = 1$) and the corresponding CDF(s) when using *Lazy*. As is seen in the figure, ACP outperforms *Lazy* and obtains a median improvement of about 100 msec in age ($\approx 33\%$ over average age obtained using *Lazy*). This over an end-to-end connection with a median RTT of about 185 msec. Further, observe that the age CDF(s) for all the sources when using either ACP or *Lazy* are similar. This hints at sources sharing the end-to-end connection in a fair manner. Also, observe from Figure 3.13b that the median RTT(s) for both ACP and *Lazy* are almost the same. This signifies that ACP maintains a backlog of update packets in a manner such that the packets don't suffer additional delays because multiple packets of the source are traversing the network at the same time.

Further, consider a comparison of the CDF of average backlogs shown in Figure 3.13c. ACP exploits the fast end-to-end connection with multiple hops very well and achieves a very high median average backlog of about 30 when using a step size of 1 and a much higher backlog when using a step size of 2. We observe that step size $\kappa = 1$ worked best age-wise. *Lazy*, however, achieves a backlog of about 1 (not shown).

For when we had 1, 2, and 5 sources sharing the WiFi access, we observe average ages per source and average RTT(s) similar to when 10 sources share the access. To exemplify, the median age obtained by an only source sending over WiFi was 190 msec ($\kappa = 1$), the median RTT was 182.5 msec, and the median backlog was 30 updates. As shown in Figure 3.13, the corresponding values for 10

sources sharing the access are 200 msec, 185 msec, and 30 updates. As the number of sources increased from 1 to 10, the average age per source wasn't impacted much by the presence of other sources. In fact, across different number of sources, each source utilized a throughput of about 1 Mbps over the end-to-end path. The sum throughput of about 10 Mbps, when there are 10 sources is far from congesting the WiFi access, which supported link rates as high as 54 Mpbs (corresponding to data payload throughputs of $\approx$ 28 Mbps). ACP optimizes age at a very low throughput of 1 Mbps, barely using a much faster WiFi link and an even faster (larger bottleneck link rate) backhaul that connects the IIIT-Delhi gateway to the server in AWS Frankfurt. We return to a similar observation, and shed more light on it in Section 3.12, where we experiment with many more sources sharing the WiFi access.

We end by showing snippets of ACP in action over the end-to-end path. Figures 3.14a and 3.14b show the time evolution of average backlog and average age, as calculated at control epochs. ACP increases backlog in small steps (see Figure 3.14a, 14 seconds onward) over a large range, followed by a rapid decrease in backlog. The increase coincides with a reduction in average age, and the rapid decrease is initiated once age increases. Also, observe that age decreases very slowly (dense regions of points low on the age curve around the 15 second mark) with an increase in backlog just before it increases rapidly. The region of slow decrease is around where, ideally, backlog must be set to keep age to a minimum.

3.10 Simulation Results

We first consider the performance of ACP during network changes. Figure 3.15 shows that ACP adapts rather quickly to the introduction of other flows that congest the network. In these simulations, we introduced one to two UDP flows at different points in the network used for simulation (Figure 3.12), where all links are 1 Mbps. ACP reduces λ appropriately and adapts backlog to desired levels.

Figure 3.16 compares the average age, source update rate λ, the RTT, and the average backlog, obtained when using ACP and *Lazy*. We vary the number of sources in the network from 1 to 20. For smaller numbers of sources, the backlog

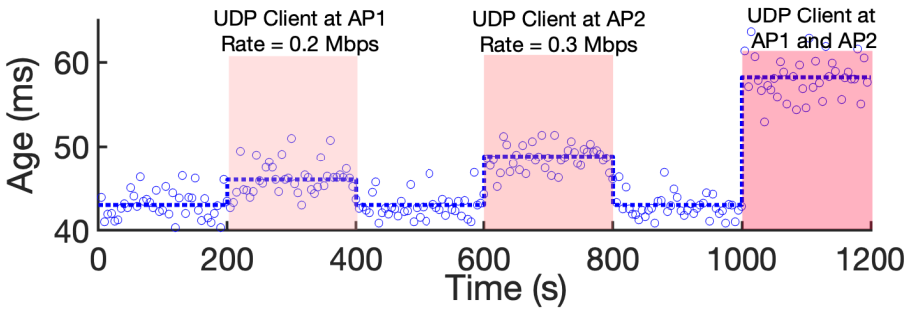
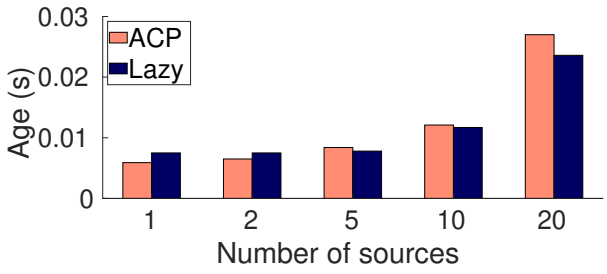
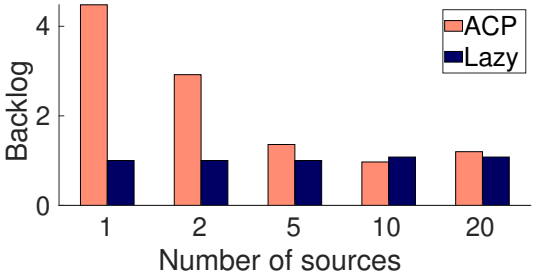


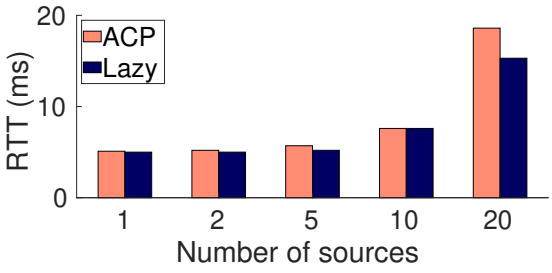
Figure 3.15: ACP adapts to network changes. Blue circles show the achieved age by an ACP client over time. A UDP client of rate 0.2 Mbps is connected to AP-1 at $200-400$ secs and $1000-1200$ secs. Another UDP client of rate 0.3 Mbps is connected to AP-2 at $600-800$ secs and $1000-1200$ secs. A darker shade of pink signifies a larger aggregate UDP load on the network.



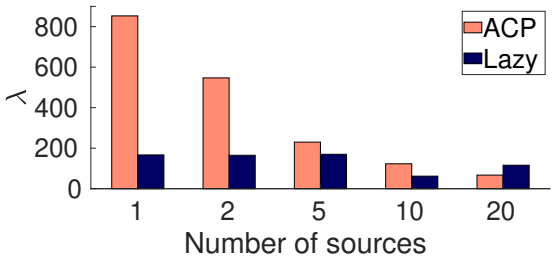
(a) Average Age



(b) Average Backlog



(c) Average RTT



(d) Average update rate λ

Figure 3.16: Comparison of ACP and *Lazy* as a function of number of sources. All sources used a WiFi PHY rate of 12 Mbps. All links other than wireless access are 6 Mbps. The sources are spread over an area of 100 m^2. The standard deviation of shadowing was set to 4 dB.

(see Figure 3.16b) per source maintained by ACP is high. This is because, given the similar rate P2P links and higher rate WiFi link, when using ACP, the sources attempt to have their update packets in the queues of the access points and routers in the network. On the other hand, a source using *Lazy* sticks to sending just one packet every RTT on an average. Thus, the average backlog per source stays similar for different numbers of sources.

As the numbers of sources become large in comparison to the number of hops (six) in the network, even at an average backlog of about 1 update per source, there is little value in a source sending more than one update per RTT. Note that there are only 6 hops (queues) in the network. When there are five or more sources, a source sending at a rate faster than 1 every RTT will have its updates waiting for each other to finish service. ACP maintains a backlog close to *Lazy* when the numbers of sources are 5 and more.

Figure 3.16d shows the average source rate of sending update packets. Observe that the average source rate drops in proportion to the number of sources. While the source rate is about 800 updates/second when there is only a single source, it is about 70 when 20 sources share the wireless access. This scaling down is further evidence of ACP adapting to the introduction of larger numbers of sources. While a source using ACP ramps down its update rate from 800 to 70, *Lazy* more or less sticks to the same update rate throughout.

The absolute improvements in average age achieved by ACP, see Figure 3.16a, for fewer numbers of sources seem nominal but must be seen in light of the fact that end-to-end RTT of the simulated network under light load conditions is very small (about 5 msec as seen in Figure 3.16c). ACP achieves a 21% and 13% reduction in age with respect to *Lazy*, respectively, for a single source and two sources.

The only impact that changing the link rates of the P2P links had was a corresponding change in RTT and Age. For example, while the average age achieved by a source using ACP in a 20 source network with P2P link rates 0.3 Mbps was ≈ 6 seconds, it was ≈ 0.25 seconds when the P2P link rates were set to 6.0 Mbps. The larger RTT for the former meant smaller λ of about 5 updates/second/source. The backlogs, as one would expect, were similar, however.

Evaluation Takeaways

As shown in Algorithm 2, ACP uses a step size parameter κ in `(INC)` and `(DEC)`. We found κ was difficult to set. For example, in our real experiments with ACP $\kappa = 1$ worked well. However, in simulations with shorter paths, with small round-trip times but not so small propagation times, a small value of $\kappa = 0.25$

ensured proper updating of the update rate λ_k.

The κ based update could also result in λ_k (calculated in Algorithm 2 line 32) becoming very small or even negative. When λ_k became negative, we reset λ_k to be at least one packet per round-trip time. Such resetting of λ_k resulted in high age in settings where multiple ACP paths shared a constrained access. While in our real-world results, we saw higher backlogs and were able to see significant improvements in average age values, ACP was not able to achieve similar gains in our simulation experiments as we increased the number of sources.

The failings of ACP borne by our extensive simulations motivate ACP+, which does significantly better than ACP and *Lazy* when a large number of sources contend over a shared access. We detail the improvements in the following section.

3.11 ACP+: An Improved Age Control Algorithm

We describe the changes in ACP+ with respect to ACP, whose control algorithm was detailed in Section 3.7. Similar to ACP, ACP+ also uses the estimate of the time average update age $\overline{\Delta}_k$ and the time average of backlog $\overline{B}_k$ at the source ACP+ at time t_k. These averages are calculated over (t_{k-1}, t_k). At every control epoch $k > 1$, at time t_k, the ACP+ source calculates the differences $\delta_k = \overline{\Delta}_k - \overline{\Delta}_{k-1}$ and $b_k = \overline{B}_k - \overline{B}_{k-1}$. However, the length $\overline{T}$ of a control epoch for ACP+ is set as $\mathcal{T} = 10/\lambda_k$. This ensures at least 10 packets are sent by the source using the updated λ_k.

The source ACP+ chooses an action u_k at the k^{th} epoch that targets a change b^*_{k+1} in average backlog over an interval of length $\mathcal{T}$ with respect to the k^{th} interval. Again the actions are broadly classified into (i) additive increase `(INC)`, (ii) additive decrease `(DEC)` and (iii) multiplicative decrease `(MDEC)`. However, unlike ACP, the κ value for ACP+ is fixed to ± 1. `MDEC` corresponds to a set of actions `MDEC`(γ), where $\gamma = \{1, 2, \ldots\}$. We have

$$\begin{aligned} &\text{INC: } b^*_{k+1} = 1, \text{ DEC: } b^*_{k+1} = -1, \\ &\text{MDEC}(\gamma)\text{: } b^*_{k+1} = -(1 - 2^{-\gamma})B_k, \end{aligned} \tag{3.2}$$

Algorithm 3 ACP+ Control Algorithm

1: **INPUT:** $b_k, \delta_k, \overline{T}, B_k$
2: **INIT:** $flag \leftarrow 0$, $\gamma \leftarrow 0$
3: **while** true **do**
4: **if** $b_k > 0$ && $\delta_k > 0$ **then**
5: **if** $flag == 1$ **then**
6: $\gamma = \gamma + 1$
7: MDEC(γ): $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
8: **else**
9: DEC: $b^*_{k+1} = -1$
10: **end if**
11: $flag \leftarrow 1$
12: **else if** $b_k > 0$ && $\delta_k < 0$ **then**
13: INC: $b^*_{k+1} = 1$
14: $flag \leftarrow 0$, $\gamma \leftarrow 0$
15: **else if** $b_k < 0$ && $\delta_k > 0$ **then**
16: INC: $b^*_{k+1} = 1$
17: $flag \leftarrow 0$, $\gamma \leftarrow 0$
18: **else if** $b_k < 0$ && $\delta_k < 0$ **then**
19: **if** $flag == 1$ && $\gamma > 0$ **then**
20: MDEC(γ): $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
21: **else**
22: DEC: $b^*_{k+1} = -1$
23: $flag \leftarrow 0$, $\gamma \leftarrow 0$
24: **end if**
25: **end if**
26: `UpdateLambda`(b^*_{k+1})
27: wait $\overline{T}$
28: **end while**

29: **function** `UpdateLambda`(b^*_{k+1})
30: $\lambda_k = \frac{1}{\overline{Z}} + \frac{b^*_{k+1}}{\overline{RTT}}$
31: **if** $\lambda_k < 0.75 * \lambda_{k-1}$ **then**
32: $\lambda_k = 0.75 * \lambda_{k-1}$ {Minimum λ threshold}
33: **else if** $\lambda_k > 1.25 * \lambda_{k-1}$ **then**
34: $\lambda_k = 1.25 * \lambda_{k-1}$ {Maximum λ threshold}
35: **end if**
36: **return** λ_k

ACP+ attempts to achieve b^*_{k+1} by setting λ_k appropriately. The estimate of $\overline{Z}$ at the source ACP+ of the average inter-update arrival time at the monitor gives us the rate $1/\overline{Z}$ at which updates sent by the source arrive at the monitor. This and λ_k allow us to estimate the average change in backlog over $\overline{RTT}$ as $(\lambda_k - (1/\overline{Z}))\overline{RTT}$. Therefore, to achieve a change of b^*_{k+1} requires choosing $\lambda_k = \frac{1}{\overline{Z}} + \frac{b^*_{k+1}}{\overline{RTT}}$ (see Algorithm 3 line 30).

Algorithm 3 summarizes how ACP+ chooses its action u_k as a function of b_k and δ_k to achieve the desired b^*_{k+1}. Please refer to Section 3.7 for details on why a particular control action is chosen.

The most significant change in ACP+ over ACP (Section 2) is in the function

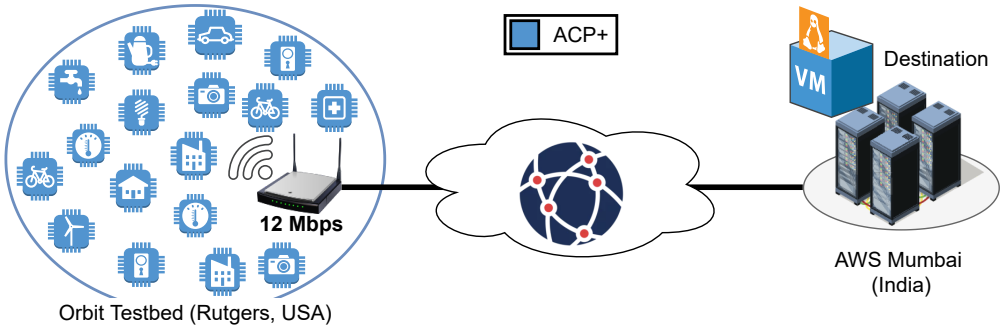


Figure 3.17: An illustration of the network topology. ACP+ clients are connected to a WiFi AP located in the Orbit Testbed's WiFi grid in USA. The server is located in AWS Mumbai, India.

`UPDATELAMBDA`. ACP uses a step size parameter κ in `(INC)` and `(DEC)`, which is difficult to set and varied for different networks. ACP+ uses a fixed $\kappa = 1$. Another difference between the algorithms is in how λ_k is set. As discussed in Section 3.7, ACP may update λ_k to a negative value that is then forced to be at least one packet per RTT. In ACP+, `UPDATELAMBDA` restricts the step change in λ and ensures that the updated λ_k is always positive. This, as shown in section 3.12, results in significant improvements in age achieved when a large number of ACP+ sources send updates over a shared multiaccess.

3.12 ACP+ Sources Update Over Intercontinental Paths Via a Contended WiFi Access

Figure 3.17 illustrates our real-world experimental setup. We used the ORBIT testbed [138], which is an open wireless network emulator grid located in Rutgers University, USA. The testbed houses multiple wireless capable and programmable radio nodes deployed in a grid fashion with a 1 m separation between adjacent nodes along columns and rows of the grid. We used the radio nodes as our sources. In addition, we configured one of the ORBIT nodes as an 802.11n access point configured to operate at 5 GHz on a fixed channel and a fixed WiFi physical layer rate using `hostapd` [139] and the `iwconfig` [140] utility. Fixed WiFi rates, in contrast to allowing WiFi rate control, enable better understanding of the impact of the WiFi access and the Internet beyond on the age of updates of the sources at the monitor. Our sources send updates to an ec2 AWS [136] instance in Mumbai,

India, which serves as our ACP+ monitor. The access point acts as a gateway to the Internet for our sources.

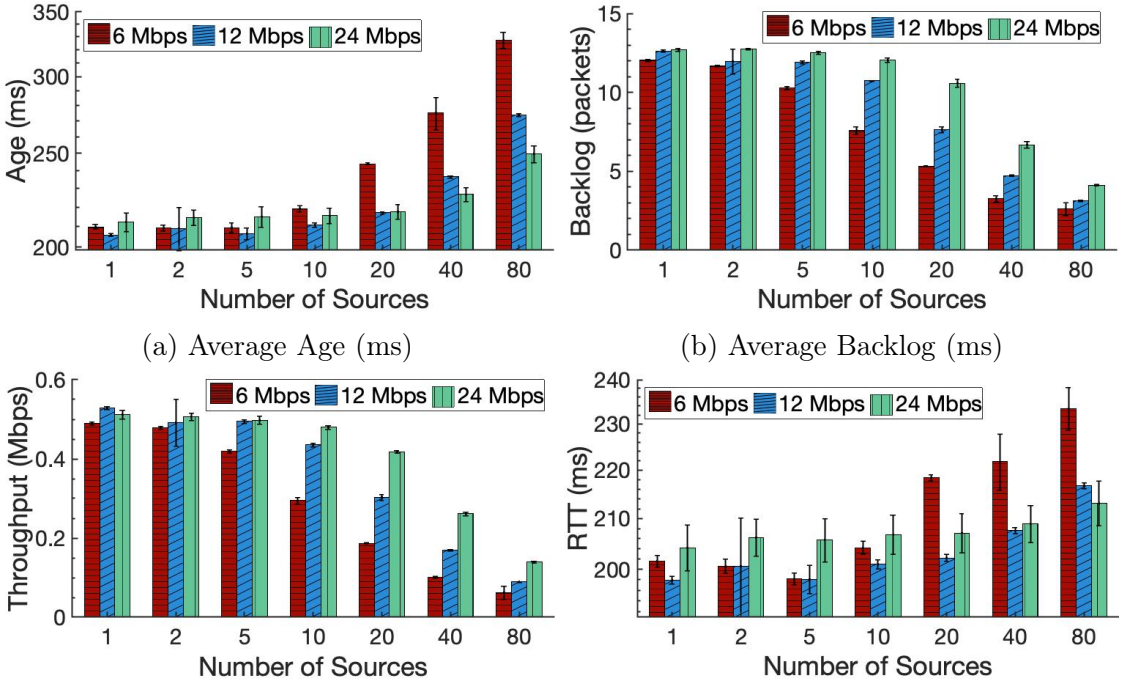
For our experiments, we selected up to 80 nodes (sources) in the testbed to connect to the WiFi access point as its clients. We also configured a node as a *sniffer* to capture packets sent over the WiFi channel. This enabled us to quantify the packet retry rates at the WiFi medium access control layer due to packet collisions or drops over the WiFi access, using the retry flag in the MAC header of sniffed packets. In the end-to-end path between the ACP+ sources in the ORBIT grid and the AWS Mumbai server, we configure only the wireless network within the ORBIT testbed. The rest of the path traverses the public shared Internet.

We experimented with 1, 2, 5, 10, 20, 40, 80 sources and WiFi physical layer rates of 6, 12, and 24 Mbps. For all our experiments, we estimated the bottleneck link rate over the end-to-end path to be the WiFi link rate. Specifically, in the absence of the WiFi access, the end-to-end path to AWS Mumbai was able to support TCP throughputs as high as 200 Mbps. Further, the *baseline* RTT between our sources and the monitor is within the 200-210 ms range. It is the average RTT observed by any source when it is the only sender to the monitor in a stop-and-wait [141] fashion. That is the source sends an update packet and waits for an `ACK` (or a timeout) before sending the next update packet. The baseline RTT is calculated in the absence of any wireless contention.

We perform at least five repeats of an experiment configuration, which includes a choice of number of sources, their locations on the ORBIT grid and the WiFi rate. Averages over the repeats are used to evaluate the performance of ACP+. Our experiments lasted over several months and were repeated over different days of the week and at different times of the day.

3.12.1 Takeaways for Age Control in the Internet

Figure 3.18c shows that ACP+ achieves a small age for a single source at an end-to-end throughput, for an update payload size of 1024 bytes, of about 0.5 Mbps, which is much smaller than our chosen WiFi rates. Note that the WiFi link rate is the bottleneck link rate for our paths between the sources and the monitor. The



(a) Average Age (ms)

(b) Average Backlog (ms)

(c) Throughput (Mbps)

(d) Average RTT (ms)

Figure 3.18: Averages of per source age, backlog, throughput, and RTT, measured over runs of ACP+ for choices of WiFi link rates and number of nodes sharing the WiFi access in the ORBIT testbed.

small age optimizing throughput was also observed in our experiments detailed in Section 3.8 for a different path in the Internet.

An age optimizing throughput much smaller than the access link rates has multiple ACP+ sources share the access without suffering an age penalty because of the other sources. Specifically, observe in Figure 3.18a that the average age per source stays similar for when there are $1-5$ sources sharing a 6 Mbps WiFi access. The age stays similar for a larger number of 10 and 20 sources when sharing, respectively, a 12 and 24 Mbps WiFi access. As the number of sources increase beyond 5, 10, and 20 sources, respectively, for access link rates of 6, 12, and 24 Mbps, the increased contention results in a rapid increase in age with the number of sources.

The increased contention results in large RTT(s) and has ACP+ maintain smaller backlogs of updates per source. The RTT(s) are shown in Figure 3.18d and the corresponding backlogs are shown in Figure 3.18b. The rapid increase in RTT per source with increasing contention, and the smaller backlogs, results in a sharp reduction in per source throughputs seen in Figure 3.18c.

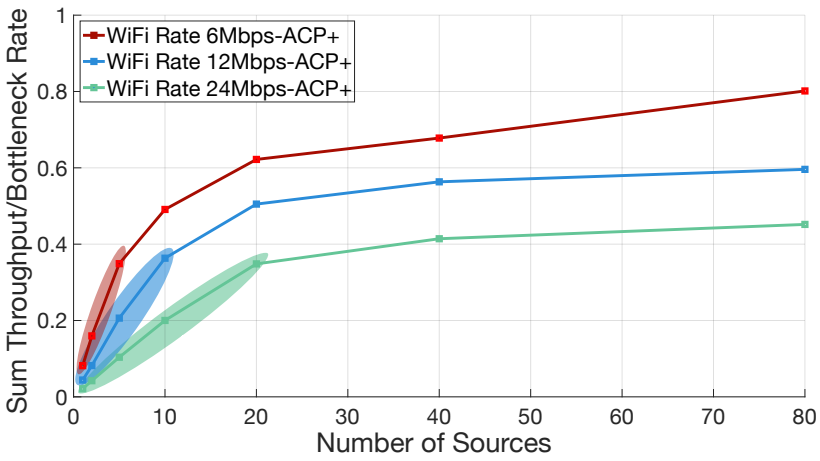


Figure 3.19: Sum Throughput of ACP+ normalized with respect to the bottleneck link. The shaded ellipse shows the region where the access is not the bottleneck for the timeliness performance of ACP+. Beyond the shaded region, the access becomes the bottleneck, and the sum throughputs start saturating.

An age optimizing throughput much smaller than access link rates has interesting consequences for age control over the Internet, as we demonstrated using ACP+. It implies that multiple sources can share the access, which is the bottleneck link in the path, without much contention and without saturating the shared access. This behavior is contrary to that of the TCP, which always saturates the bottleneck link, irrespective of the number of senders sharing it.

For a small enough number of sources sharing an access, age optimization is constrained by the backhaul beyond the access. While the backhaul has a bottleneck link rate much larger than that of the WiFi access, the time that an update spends buffered in the many hops that constitute the backhaul, given the other traffic using it, is the most salient as regards age control. As the number of sources sharing an access becomes large, the access link is saturated and becomes the constraining factor with regards to age optimization.

Figure 3.19 illustrates the two regimes of age control. It plots the utilization of the access (measured as the sum of throughputs of the sources sharing it) normalized by the WiFi link rate. For when the number of sources sharing the access is small enough such that contention between the sources is low, the normalized utilization of the access increases in proportion (marked by the shaded ellipses) to the number of sources. The access isn't the constraining factor and age control

must adapt to the utilization of the backhaul by other traffic, while ensuring that a large enough backlog of source updates is maintained in the backhaul, given the large number of hops that may constitute it. Beyond the region of low contention, the normalized utilization flattens and converges close to the maximum obtainable for the link rate[3]. Not surprisingly, the region of low contention extends to a larger number of sources for a larger WiFi link rate.

We could not extend our experiments to include greater than 80 nodes on the ORBIT testbed due to hardware and wireless driver restrictions. As a result, we couldn't explore the region of really high contention wherein one would expect age control to ideally backlog on an average less than an update per source per round-trip-time. Recall that ACP's failing was its inability to do better than *Lazy* when there was high contention. Like *Lazy*, ACP maintained backlogs of about 1 per source as contention increased. We resort to simulations to evaluate how ACP+ performs with respect to *Lazy*, especially when the contention resulting from the sources sharing the access is very high.

3.13 Simulations Setup and Results

We used the same simulation setup as discussed in Section 3.8 using network simulator ns3[4] together with the YansWiFiPhyHelper[5]. The base network topology used in our simulations is shown in Figure 3.12. We compare the performance of ACP+ to *Lazy*, which as mentioned earlier, is a conservative status updating mechanism that sends one update per $\overline{\text{RTT}}$ and maintains an average backlog of 1 update in the network for any source.

We show results for when source nodes are spread uniformly and randomly over an area of 20×20 m^2. We chose the number of sources from the set $\{1, 6, 12, 24, 48\}$. The channel between the source and AP-1 was log-normally distributed with a standard deviation of 12 and a path loss exponent of 3. The WiFi link rate was set to 12 Mbps and that of the P2P links was set to 6 Mbps.

[3]The normalized utilization is less than 1 because of WiFi protocol overheads, including headers of layers 1 and 2. The overheads are larger for larger link rates.

[4]https://www.nsnam.org/

[5]https://www.nsnam.org/doxygen/classns3_1_1_yans_wifi_phy.html

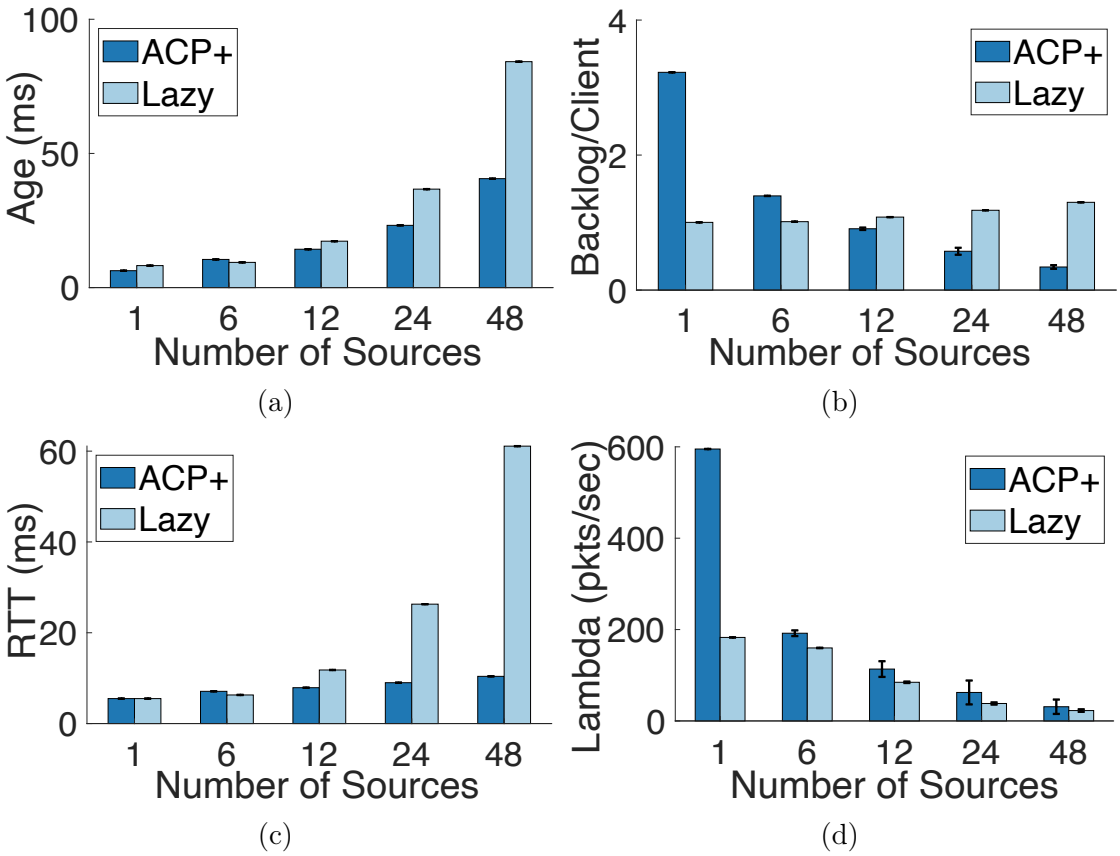


Figure 3.20: (a) Average source age (b) Average source backlog (c) Average source RTT and (d) Update rate λ for *Lazy* and ACP+ when all links other than wireless access are 6 Mbps. All sources used a WiFi PHY rate of 12 Mbps. The sources are spread over an area of 400 m^2.

Figure 3.20a shows that ACP+ achieves a smaller age per source than *Lazy*. The improvements are especially significant when a large number of sources share the access to AP-1. That ACP+ is able to achieve smaller ages can be understood via the average backlog per source when using ACP+ and *Lazy*, which is shown in Figure 3.20b. When we have just one source, ACP+ tries to fill each queue in the network with an update. This results in a larger backlog and a lower age in comparison to *Lazy*, which achieves a backlog of just 1 update. However, as the number of sources increases, while *Lazy* continues to maintain a backlog of 1 per source, ACP+ reduces it. The backlogs obtained are $3.23, 1.39, 0.91, 0.57, 0.34$, respectively, for $1, 6, 12, 24, 48$ sources. Compare these backlogs with our earlier simulation results using ACP in Figure 3.16b. ACP's design choices forced it to maintain a minimum backlog of 1 per source. This resulted in *Lazy* doing at least as well as ACP for a large enough number of sources connected to the shared access (see Figure 3.16a).

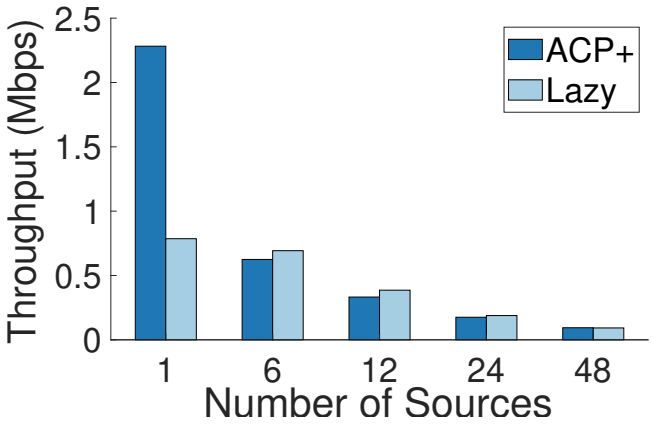


Figure 3.21: Average per-source Throughput (Mbps)

The ACP+ backlogs when we have a large number of sources are not only much smaller than *Lazy*, it turns out that they are not too far from an ideal scheduling mechanism that schedules updates from the sources in a round-robin manner. For simplicity, ignore the difference in the link rates of the WiFi and P2P links. Also, assume that no packets are dropped due to channel errors over WiFi. A round-robin scheduler would keep six updates in transit of the source when we have just one source. This would result in a backlog of 6. It would schedule six sources one after the other in a manner such that a round of scheduling would lead to six packets in the six queues from the six different sources, resulting in an average backlog of 1 per source. Similarly, for when we have $12, 24, 48$ sources, we would see backlogs per source of $1/2, 1/4, 1/8$, respectively. ACP+ sees larger backlogs than these, at least partly because of packet collisions over the WiFi access, which results in larger delays in the WiFi hop.

ACP+'s good adaptation to an increase in the number of sources is also seen in the fact that the RTT doesn't increase much as the number of sources increase. This is unlike *Lazy* which sees big increases in RTT. While ACP+ results in RTT of $5.5, 7.1, 7.9, 9, 10.4$ ms, respectively, for $1, 6, 12, 24, 48$ sources, *Lazy* sees RTT of $5.5, 6.3, 11.8, 26.3, 61.1$ ms.

Figure 3.21 shows the throughput per source when using ACP+ and *Lazy*. Consider ACP+. The throughput for when there is only one source is ≈ 2.25 Mbps. The source ACP+ attempts to fill the bottleneck link in the simulation setup. In the absence of other traffic flowing over the WiFi and P2P links, having an average of 1 update per P2P link is age optimal. Given the 5 p2p links that are

6 Mbps each and a WiFi access that has a link rate of 12 Mbps, one would expect an average backlog of 5.5 packets at the rate λ of sending updates that minimizes age. ACP+ maintains an average backlog of a bit more than 3 packets. While less than optimal, the reduction in average age when going from a backlog of about 3 to that of 5.5 is nominal. Specifically, the minimum age is empirically determined to be $\approx$ 5.69 ms, while ACP+ obtains an age of 6.3 ms. As the number of sources increases, the throughput per source reduces. The total utilization (sum throughput) quickly approaches the bottleneck link rate, though. Six sources utilize about 4 Mbps together and for 50 sources, the total utilization is about 4.5 Mbps.

Sources using *Lazy* see about the same per source throughput as when using ACP+ when there are 6 or more sources. However, the backlog and RTT is larger when using *Lazy*, resulting in a larger age.

3.14 Age Fairness Using ACP+

We quantify fairness in age achieved by multiple ACP+ sources that share an access network and send their updates to a monitor. We use the Jain's fairness index [142] to quantify *age fairness* in both our simulations and real-world experiments. The Jain's fairness index can take values between 0 and 1. A larger fairness index implies more similar ages of the different sources at the monitor. An index of 1.0 indicates that the ACP+ control algorithm enables the ACP+ flows to achieve the same ages over their paths to the monitor.

In our simulations, we find that as we increase the number of sources sharing the network from 6 to 48, our fairness index reduces from .99 to .89. In our real-world experiments, the fairness index lies between .99 to 1.0 as we increase source density from 2 to 80 for all the WiFi link rates (6, 12 and 24 Mbps). ACP+ ensures age fairness in our experiments.

3.15 Chapter Summary

We motivated the need to optimize the freshness of updates generated by a source, at a monitor that receives the updates over a network. Having shown the drawbacks of using TCP to have sources send updates to their monitors, we proposed a protocol stack with a transport layer protocol, namely the age control protocol, that sends and receives its packets over UDP. We detailed the control algorithm of ACP and the improvements over it that constitute ACP+. We quantified the performance of the control algorithms over end-to-end paths that connect IoT devices to the cloud via extensive experimentation using simulations and real-world networks. Our experiments helped characterize salient features of age control over the Internet.

In this chapter, we focused on the age control protocol as an end-to-end transport mechanism for sending update packets to a server over the Internet. In the next chapter, we will detail its differences vis-a-vis different TCP congestion control mechanisms and their ability to transport fresh updates over an end-to-end path consisting of both core and access networks.

Chapter 4

Congestion Control and Ageing in the Internet

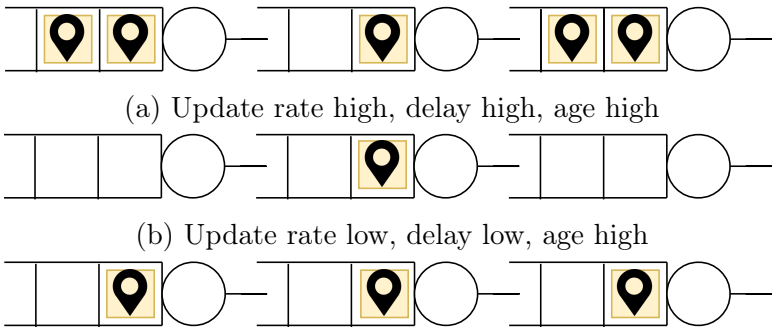
4.1 Introduction

The challenge of age control over an end-to-end path in the Internet is to adapt the rate of status updates entering the path so that there are as many status updates in transit as possible while no update waits behind another in a router queue. This is in contrast to TCP loss-based congestion control algorithms that aim for high throughput by having as many packets as possible in each queue without exceeding the available queue occupancy. This allows an end-to-end flow to achieve a rate equal to the bottleneck bandwidth; however, this is at the expense of large delays and eventual losses due to excessive queueing at the bottleneck.

Recently, requirements of low latency along with high throughput have led to the proposal of hybrid congestion control mechanisms such as BBR [32]. At its stated ideal point of operation, BBR would have TCP packets delivered to the receiver at the bottleneck link rate, while each packet would experience an average delay as that experienced by a packet if only it was sent over the path. Intriguingly, this would satisfy the goal of age control by resulting in the highest rate of packet delivery at the receiver at the smallest possible packet delays.

In the previous chapter, we proposed and presented a strong case for the Age Control Protocol, a transport layer protocol that regulates the rate at which status updates are sent by an application over an end-to-end path. By abstracting away an end-to-end path as a series of queues, we argued that a good age control algorithm must try to have as many status updates in transit as possible while ensuring that the updates don't wait for previously queued prior updates from the application.

Figure 4.1 provides an illustration of a good age control strategy in action for an end-to-end path of three identical queues, each with deterministic service times. Figures 4.1a and 4.1b, respectively, have too many and too few updates,



(a) Update rate high, delay high, age high

(b) Update rate low, delay low, age high

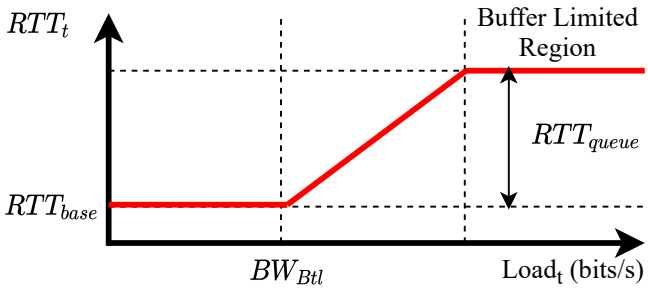
(c) Ideal snapshot of updates in transit

Figure 4.1: An illustration of queue occupancy and its impact on age.

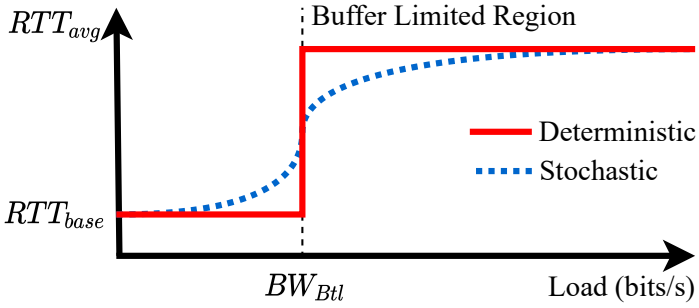
resulting in high age. Figure 4.1c shows the snapshot one would expect to see with a good age control algorithm sending updates over the three-queue network. Of course, the picture becomes more complicated when the queues have random service times. For example, with a pair of M/M/1 queues in tandem, the average number of packets queued in the system at minimum age was shown to be ≈ 1.6 updates (See Figure 3.7).

While applications have diversified significantly over the past few decades, TCP is still the dominant protocol used in the Internet with $\approx 90\%$ traffic share [143]. TCP congestion control is the primary mechanism by which end hosts share available Internet bandwidth. For the purpose of TCP's operation, the end-to-end path may be abstracted away as a link with bottleneck bandwidth BW_{Btl} and a round-trip propagation time of RTT_{base} (baseline RTT) [32]. Figure 4.2a provides an illustration, akin to that in [32, Figure 1], of the *instantaneous* round-trip time RTT_t at time t as a function of the current offered load (the effective rate at which TCP is sending bytes). As long as the offered load is smaller than BW_{Btl}, the TCP packets see a low RTT of RTT_{base}. Once the offered load becomes larger than BW_{Btl}, the TCP packets that arrive at the link's queue see increasingly more packets waiting for service ahead of them. This results in a linear increase in RTT_t until the queue becomes buffer limited, the RTT saturates and TCP packets arriving at a full queue are dropped.

Traditionally, TCP's congestion control allows for an increasing number of unacknowledged bytes from an application to flow through the network pipe until one or more bytes are lost due to the resulting congestion. Such a loss-based



(a) RTT as a function of offered load



(b) Average steady state RTT as a function of average load

Figure 4.2: An illustration of how round-trip times vary as a function of the offered load. While (a) shows the change in instantaneous RTT as the load increases, (b) shows the steady-state average behavior at a chosen load.

congestion control algorithm keeps increasing the offered load until a packet is lost as a result of the link operating in the buffer limited region. The flow will achieve a throughput equal to the bottleneck bandwidth, but packets in the flow will suffer large round-trip times, especially when the link has a large buffer.

Figure 4.2a suggests that one would like to operate at the lower "knee" in the curve, i.e., close to the bottleneck throughput BW_{Btl} at low delays. In fact, delay-based and hybrid congestion control algorithms such as the recently proposed *Bottleneck Bandwidth and Round-trip propagation time* (BBR) protocol, attempt this by using the round-trip time to detect congestion early before a loss occurs due to buffer unavailability at a certain router along the path. Note that this combination of a throughput of BW_{Btl} and round-trip times of RTT_{base} is in fact achieved by the snapshot in Figure 4.1c that illustrates a good age control algorithm in action.

Of course, as was observed in [144] in relation to the stated point of operation of BBR, when a path is better modeled by a stochastic service facility, the average round-trip times at the maximum achievable throughput of BW_{Btl} could

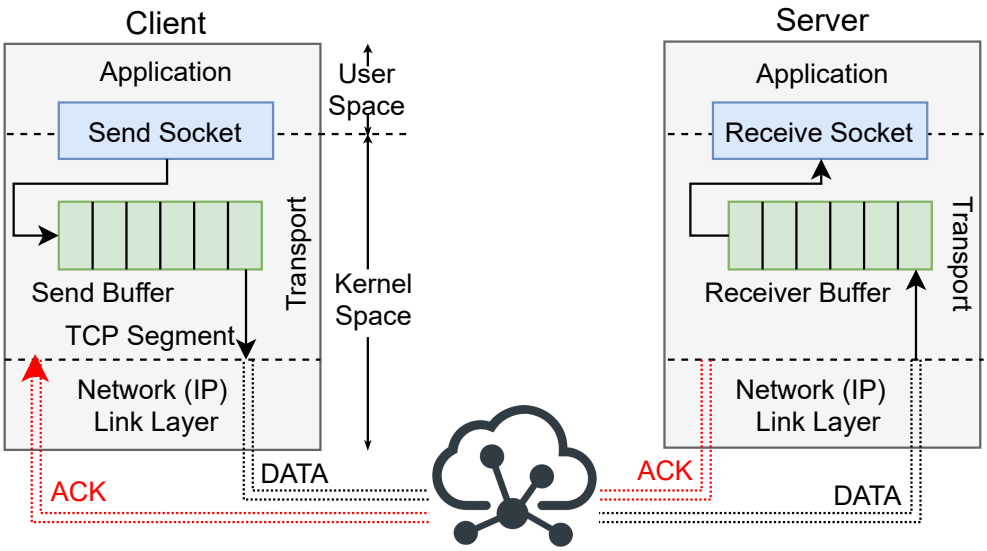


Figure 4.3: TCP Network Stack

be much larger than RTT_{base}. Figure 4.2b provides an illustration of *steady-state* average RTT as a function of average load. The red and blue curves, respectively, correspond to a deterministic and a stochastic service facility.

This shift in congestion control algorithms from keeping the pipe full to "keeping the pipe just full, but no fuller" [144], motivates this empirical study of how the information at a receiver would age if updates were transmitted over the cloud using the congestion control algorithms. However, we must be careful as (a) TCP doesn't regulate the rate of generation of packets by the status updating application, (b) it is a stream-based protocol and has no notion of update packets. As illustrated in Figure 4.3, an application writes a stream of bytes to the TCP sender's buffer. TCP creates segments from these bytes in a first-come-first-serve manner. TCP segments are delivered to the TCP receiver. At any time, TCP allows a total of up to a current congestion window size of bytes to be in transit in the network. The TCP receiver sends an `ACK` to inform the sender of the last segment received.

To stay focused on evaluating how scheduling TCP segments over an end-to-end path would age updates at a receiver, we assume that a TCP segment, when created, contains fresh information. Specifically, we ignore the ageing of bytes while they wait in the TCP send buffer. One way of achieving this in practice would be to have the application provide freshly generated information (as in a generate-at-will model [77]) to be incorporated in a TCP segment just as TCP

schedules it for sending.

We approximate the age of the segment when it arrives at the TCP receiver to be the RTT of the segment, which is calculated based on the time of receipt of the TCP `ACK` that acknowledges receipt of the segment. Further, we approximate the inter-delivery time of segments at the receiver by the inter-delivery times of the corresponding `ACK`s. The RTT(s) and the inter-delivery times together allow us to come up with an estimate of the time-average of age at the receiver that results from a chosen congestion control algorithm, using the graphical method of time-average age calculation using the age sample function similar to one shown in Figure 3.6 [2].

Last but not the least, we would like to minimize the impact of packet loss due to link transmission errors on our evaluation of congestion control. Given our focus on paths in the cloud, specifically between AWS data centers, we observe a very small percentage of loss, and that too because of buffer overflows in routers that result in the process of congestion control estimating the bottleneck bandwidth.

Our specific contributions of this chapter are:

1. We provide an empirical study of age, throughput and delay trade-offs obtained when using state-of-the-art TCP congestion control algorithms to transport updates over an end-to-end path in the cloud.

2. We evaluate a mix of loss-based (Reno [145] and CUBIC [146]), delay-based (Vegas [147]) and hybrid congestion control algorithms (YeAH [148] and BBR [32]) for different settings of receiver buffer size.

3. We compare the performance of the TCP algorithms with that of ACP+. We show that ACP+ does well in estimating the network conditions on the end-to-end path and appropriately adapts the rate of status updates sent over the path to keep age at the receiver close to the minimum. (Section 4.4)

4. We also compare ageing of update packets in shared and contended access networks with up to 80 nodes on the ORBIT grid at Rutgers University. We find that as contention on the access network increases, the performance of TCP degrades considerably. In high contention networks, ACP+ performs 60× better *age-wise* compared to the best performing TCP variant (Section 4.6).

4.2 Primer on TCP Congestion Control

Congestion control was introduced in the Internet in the 1980s to overcome the congestion collapse of the Internet [15]. These algorithms help achieve higher utilization of the available network bandwidth while avoiding overloading the network. Over the years, the design strategies of congestion control have evolved from loss-based to delay-based and now recently introduced hybrid strategies to improve the TCP's performance in different networks such as lossy wireless or in the presence of large network buffers.

The **loss-based** strategies aim for high throughput by having as many packets as possible in each queue without exceeding the available queue occupancy. This allows an end-to-end flow to achieve a rate equal to the bottleneck bandwidth; however, this is at the expense of large delays and eventual losses due to excessive queueing at the bottleneck. These algorithms generally have a slow start phase followed by a congestion avoidance phase. A few examples of loss-based congestion control algorithms are Reno, NewReno [149], Highspeed-TCP [150], Scalable TCP [151], TCP Westwood [152], TCP Westwood+ [153], BIC [154] and CUBIC [146]. CUBIC is the current default congestion control algorithm in the Linux kernel. CUBIC differs from the traditional congestion control algorithms like Reno/NewReno in setting the congestion window size as a cubic function. It employs a fast recovery after the loss event and is less aggressive as it approaches the `cwnd_max`, making it suitable for flows that require higher bandwidth. CUBIC offers high RTT fairness, where RTT fairness is a fairness measure among connections with different RTTs sharing the link. This is because the `cwnd` calculation is independent of RTT. However, it fails to achieve maximum available bandwidth utilization and leads to packet losses due to its congestion detection mechanism.

The **delay-based** strategies monitor the delay over the Internet using the RTT measurements and aim to keep the queueing delays below a certain threshold and avoid the large delays associated with the loss-based approaches. These are proactive schemes instead of reactive loss-based approaches. The use of delay (instead of losses) as a congestion indicator helps avoid queue buildup, which in turn is advantageous for low-latency applications. Additionally, these mechanisms avoid

the throughput oscillations associated with the Additive Increase Multiplicative Decrease (AIMD) based approaches such as CUBIC. However, RTT-based congestion approaches suffer due to delayed `ACK`s, cross-traffic and queues in the network. A few examples of these are Vegas [147], FAST [155], TCP LoLa [156] and TIMELY [157]. Vegas uses RTT measurements to estimate the buffer occupancy at the bottleneck queue as a function of expected and actual transmission rate and attempts to keep it under a predefined threshold [158]. RTT_{base} is used as a baseline measurement for a congestion-free network. The theoretical maximum expected transmission rate in a congestion-free network is $\texttt{cwnd}/RTT_{\texttt{base}}$. A Vegas flow will achieve this rate if all transmitted packets are acknowledged within the minimum RTT such that RTT_{i}=RTT_{base}, where RTT_{i} is the RTT of i^{th} packet. Similarly, the actual transmitted rate is calculated using the currently measured RTT as $\texttt{cwnd}/RTT_{\texttt{i}}$. Therefore the number of packets queued at the bottleneck can be calculated as

$$\Delta = \texttt{cwnd}\frac{RTT_{\text{i}} - RTT_{\text{base}}}{RTT_{\text{i}}}$$

At reception of each `ACK`, Vegas calculates Δ and keeps it between predefined thresholds α and β. For Linux kernel, Δ values lies between $2-4$. A Δ value higher than β indicates congestion and the `cwnd` is reduced by one. If Δ is lower than α, `cwnd` is increased by one.

Vegas minimizes the queueing delays and reduce throughput oscillations, resulting in improvements in long-term throughput averages. Despite having these benefits, Vegas suffers from some inherent issues. First, the algorithm has a slow growth rate and, therefore, can lead to the under-utilization of resources in a high-speed network. Secondly, the algorithm uses RTT measurements to calculate the sending rate. Any change in the network leading to an increase in RTT is interpreted as congestion and results in an unnecessary reduction in the sending rate. Finally, it doesn't work well when other loss-based flows are sharing the network and switches to a loss-based approach in this scenario.

Hybrid algorithms use the combination of loss and delay as congestion indicators. While the loss-based strategies work well in a high-speed network with low resource utilization as these can quickly ramp up the `cwnd` sizes, the delay-based schemes are more suited for higher utilization congested networks. The basic

idea behind the hybrid approach was to improve on the loss-based strategies such that congestion is detected much before the queues build up and the packets are dropped yet keeping the throughput and network utilization high. A few examples include TCP Compound [159], Veno [160], Illinois [161] and YeAH [148].

In 2016, Google proposed the **bottleneck bandwidth and round-trip time (BBR)** [32] congestion control algorithm, which aims to utilize available network bandwidth without filling network pipes. BBR periodically estimates available network bandwidth (BW_{Btl}) using the maximum data delivery rate and baseline round-trip time (RTT_{base}). As shown in Figure 4.2, working at this point can ensure maximum data delivery rate with minimum congestion in a deterministic network environment. BBR uses four different phases for its operation, namely `startup`, `drain`, `probe bandwidth` and `probe RTT`. The `startup` phase uses the slow start mechanisms of loss-based algorithms, essentially doubling the sending rate every RTT. BBR uses the received `ACK`s to estimate the current delivery rate. BBR assumes that the bottleneck rate is achieved when the delivery rate stops increasing for *three* consecutive RTTs. This marks the end of `startup` phase and BBR enters the `drain` phase to remove excess queue buildup in the previous sending cycle. BBR reduces its sending rate to $0.75 \times BW_{\text{Btl}} \times RTT_{\text{base}}$. BBR calculates the RTT of the every received `ACK` and sets RTT_{base} as the minimum value of RTT in the past 10 seconds. The `cwnd` is set to the bandwidth delay product (BDP).

$$\texttt{cwnd} = BW_{\text{Btl}} \times RTT_{\text{base}}.$$

To adapt to the changing network conditions, BBR periodically launches the `probe bandwidth` phase by sending data at a higher rate ($1.25 \times BDP$) for an RTT interval. This is followed by a new `drain` phase where the rate is reduced to drain the excess queues formed. This is done once every eight cycles each lasting for RTT_{base} duration. If the RTT_{base} is same for past 10 seconds, BBR enters the `probe RTT` phase and reduces the in-flight data to four packets in order to drain the bottleneck queue entirely. This phase lasts for 200 ms. BBR is designed to spend majority of time in `probe bandwidth` with `probe RTT` being triggered for $\approx 200ms$ once every 10 seconds.

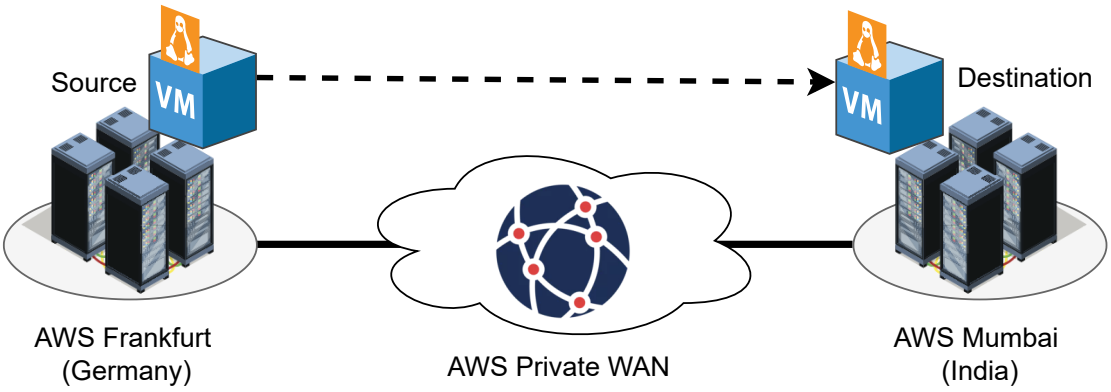


Figure 4.4: An illustration of the real experiment topology on the AWS ec2 cloud network. The client machine (both ACP+/TCP) was in AWS Frankfurt, Germany, and the server was in AWS Mumbai, India. The instances were connected via the AWS Private WAN.

4.3 Ageing over the Internet

We empirically determine the ability of TCP congestion control algorithms to deliver *fresh* updates over an end-to-end Internet path. We also compare its performance with ACP+. First, we focus on understanding how ACP+ and TCP behave in the Core Network (See illustration in Figure 3.1). The Internet core is widely regarded to be significantly reliable (as also seen in our experiments described later) and is operated by managed entities such as Amazon Web Services (AWS). Further, we investigate the behavior of the algorithms over an end-to-end path when the first hop is a shared and contended wireless access (Figure 3.1), wherein the differences between age control and TCP congestion control for maximizing throughput over the Internet are the most stark and provide valuable insights.

4.4 Ageing over the Core Network

To understand the behavior of ageing when using different transport protocols, we conduct real-world experiments over the AWS cloud network.

Setup and Methodology: Figure 4.4 shows the real experiment topology. All our experiments over the Internet used two T2.micro instances in the AWS ec2 cloud network. Both instances are configured with one virtual CPU, 1 GB RAM and a 1 Gbps Ethernet link connected to the AWS private WAN. One of the

instances was in the AWS Frankfurt (Germany) data center, while the other was deployed in the AWS Mumbai (India) data center. Each instance ran a virtual machine with Ubuntu 18.04 LTS with Linux kernel version 5.3. We confirmed through periodic `traceroute` that the underlying network between our two chosen instances was served by the AWS private WAN.

We describe our measurement methodology next. For both the ACP+ and TCP experiments, we deployed the sender in AWS Frankfurt and the receiver in AWS Mumbai. For each chosen congestion control algorithm, we investigated the impact of different receive buffer sizes on the performance of the congestion control algorithms by changing `default` and `maximum` values of `r_mem` in the Linux kernel. The space available in the receiver buffer limits the maximum amount of bytes that any congestion control algorithm may send to the TCP receiver.

For the TCP experiments, we used `iPerf3` for packet generation and `Wireshark` for packet captures. To ensure that all algorithms saw similar network conditions, we ran multiple iterations of ACP+, TCP BBR, TCP CUBIC, TCP Reno, TCP Vegas and TCP YeAH, in that exact order, one after the other. For each TCP variant in the stated order, we further ran different receive buffer settings. Each run of the experiment lasted 200 s. Considering that end-to-end RTT is $\approx$ 110 ms in our setup, TCP spends a majority of the transfer time in the steady-state phase.

TCP guarantees delivery of bytes sent by an application. As a result, TCP retransmits lost segments, which might contain stale information. We ignore retransmitted segments for calculation of the time-average age. However, retransmissions and delays incurred because TCP ensures in-order delivery of bytes to the receiving application can result in a large age. Since our goal is to understand the behavior of congestion control algorithms, we would like to minimize the impact of guaranteed in-order delivery on age achieved by the algorithms. Luckily, the core network provides a very reliable byte pipe. Even our measurements over the controlled wireless testbed, which we detail later, saw very few losses ($< 0.1\%$) for up to 80 nodes. We also measured the duplicate `ACK`s, which indicate losses and out-of-order packets received at the receiver. We observed very low percentages of duplicate `ACK`s ($< 1\%$) for up to 20 nodes and up to $\approx 1.5\%$ for up to 80

nodes. We matched the time of reception of dupACKs to retransmitted data to establish a correspondence. Similar to our duplicate `ACK`s, the retransmitted data percentages are very low ($\approx 0.5\%$) for up to 80 nodes. Please note that these are percentages and not fractions. Given the low percentages of retransmissions, we observe that the age performance of TCP congestion control algorithms does not take a hit because of TCP's feature of guaranteed in-order delivery.

4.5 Results over the Core Network

We show results from 40 runs each of ACP+, BBR-d1m1, BBR-d1m3, BBR-d5m5, CUBIC, Reno, Vegas and YeAH. For each run, we show the average age, throughput, and average delay (round-trip time). In the above list, we have BBR run with three different receiver buffer settings. BBR-d1m1 denotes the smallest `default` and `maximum` values of the receiver buffer (`r_mem`). In BBR-d1m3, the `default` is the same as BBR-d1m1 but the `maximum` is three times larger. Similarly, in BBR-d5m5 both the `default` and the `maximum` is five times that in BBR-d1m1. For all other TCP algorithms, the results are shown for a `default` and a `maximum` five times that of BBR-d1m1. In general, one would expect a larger receiver buffer to allow the TCP algorithm to have a larger number of bytes in flight as long as the network doesn't become the bottleneck.

4.5.1 Queue Waiting Delays Dominate

Figure 4.5 shows the impact of TCP segment lengths on delay. As is seen, segment length and delays are uncorrelated for all the TCP algorithms. This observation can be explained by the fact that the delays in the network are almost entirely because of the time spent in router queues awaiting transmission. The transmission times (propagation delays), which are about 20 ms, are a small fraction in comparison. It may be worth noting that the TCP segment lengths are chosen by the TCP algorithm and often change during a TCP session. In the figure, we show segment lengths averaged over a run.

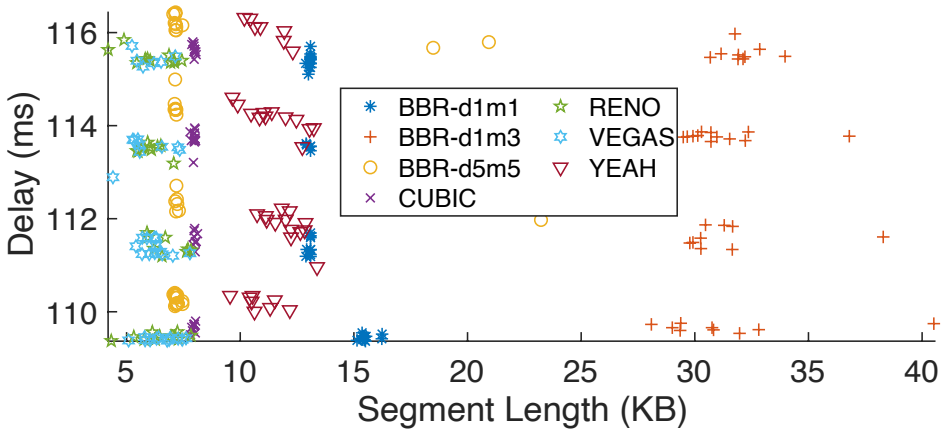


Figure 4.5: TCP segment length vs. delay obtained for the runs of the different algorithms.

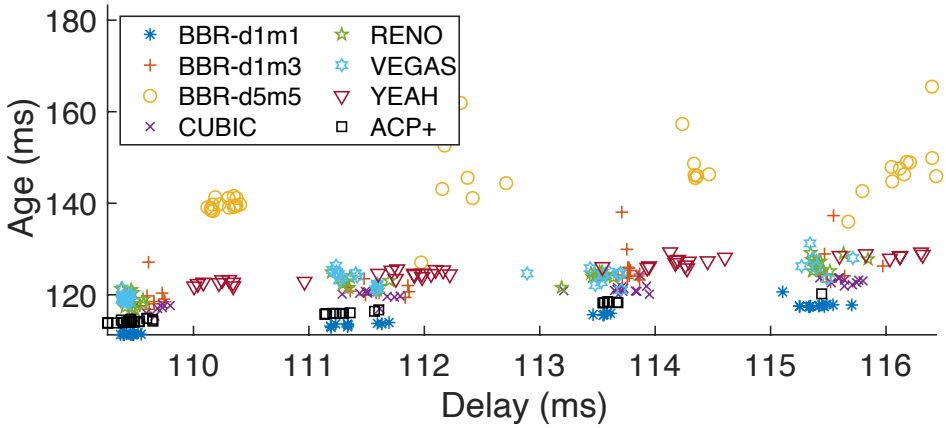


Figure 4.6: Delay vs. age for the different runs of the chosen algorithms.

4.5.2 Delay vs. Age

Figure 4.6 shows a scatter of (delay, age) for the chosen runs. We see that BBR-d5m5 sees both age and delays larger than the rest. Amongst the rest, from the figure, it is apparent that ACP+ achieves delays and ages smaller than all algorithms other than BBR-d1m1. BBR-d1m1 achieves a slightly smaller age than ACP+.

In fact, the age and delay achieved by BBR-d1m1, averaged over all runs, are 114.5 ms and 112.33 ms, respectively. The corresponding values for ACP+ are 115.5 ms and 110.79 ms. The next smallest age is achieved by CUBIC and is $\approx$ 121 ms. Reno, Vegas and BBR-d1m3 achieve higher ages than CUBIC, with YeAH achieving the highest age of about 125 ms among them. BBR-d1m4, BBR-d1m5 and BBR-d5m5 achieve ages larger than 140 ms. Only BBR-d5m5 is shown.

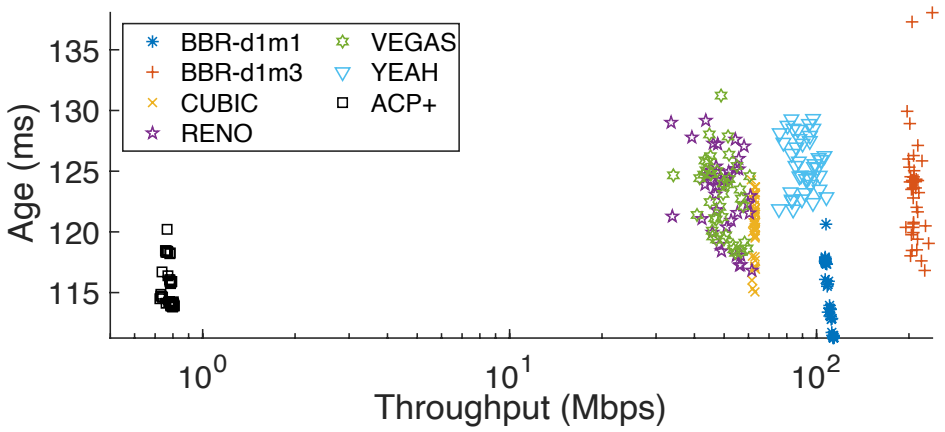


Figure 4.7: Throughput vs. age for the different runs of the chosen algorithms.

4.5.3 ACP+ vs. BBR-d1m1

Before we delve further into the relative performances of ACP+ and BBR-d1m1, let's consider Figure 4.7 in which we show the (throughput, age) values achieved by the different algorithms. We omit BBR-d5m5 from the figure as it resulted in high age values (average larger than 140 ms) and also did not yield very good throughput. BBR-d1m3 achieves the highest throughput. In fact, its throughput of about 200 Mbps is twice the next highest value of about 110 Mbps achieved by BBR-d1m1. The average age when using BBR-d1m3 is 123.5 ms in contrast to the 114.5 ms obtained when using BBR-d1m1.

Interestingly, the throughput obtained by ACP+ is a low of 0.77 Mbps in contrast to 110 Mbps obtained using BBR-d1m1 ($\approx 141\times$ the ACP+ throughput). This stark difference is partly explained by the segment[1] sizes used by BBR-d1m1, on an average about 14 KB, in comparison to the constant 1024 byte payload of an ACP+ packet. This difference still leaves an unexplained factor of about 10. This is explained by an average inter-ACK time of 10.4 ms for ACP+ in comparison to a much smaller 1.16 ms for BBR-d1m1 that results from BBR-d1m1 attempting to achieve high throughputs.

To summarize, ACP+ results in an average age of 115.5 ms, an average delay of 110.79 ms, an average throughput of 0.77 Mbps and an inter-ACK time of 10.4 ms. The corresponding values for BBR-d1m1 are 114.5 ms, 112.33 ms, 110 Mbps and 1.16 ms. *ACP+ achieves an almost similar age as BBR-d1m1, however, at a*

[1]Recall our assumption that every new segment contains a fresh update.

significantly lower throughput. The similar age at a much larger inter-ACK time is explained by the fact (observed in our experiments) that while a very low or high rate of updates results in high age, age stays relatively flat in response to a large range of update rates in between. It turns out that ACP+ tends to settle in the flat region closer to where increasing the rate of updates stops reducing age. This much reduced throughput of ACP+ is especially significant in the context of shared access, allowing a larger number of end-to-end ACP+ flows to share an access without it becoming a bottleneck.

4.5.4 The BBR Puzzle

What could explain the low age achieved by BBR-d1m1? We observe that the average delay of 112.33 ms when using BBR-d1m1 is the same as that obtained by a *Lazy* (introduced in [28]) status updating protocol we ran alongside the others, which sends an update once every round-trip time. One would expect *Lazy* to achieve a round-trip time of RTT_{base} (see Figure 4.2a). This tells us that BBR-d1m1's flow on an average saw an RTT of RTT_{base}. While it obtained a low throughput of 100 Mbps, it seems to have kept the pipe full enough. This low throughput was an accidental consequence of the receiver buffer size settings of BBR-d1m1, which disallowed the congestion control algorithm to push bytes into the network at a larger rate. The higher throughput achieved by BBR-d1m3, as observed earlier, came with a higher age, however.

4.6 Age over Shared and Contended Access

Figure 3.17 (Chapter 3) illustrates our real-world experimental setup to evaluate ACP+ and TCP for delivery of *fresh* updates in a shared access network. We use the same setup and methodology as discussed in Section 3.12 of Chapter 3 for our experiments.

The baseline RTT (RTT_{base}) between our sources and the monitor is 200-210 ms. Our experiment setup allows us to analyze a typical IoT environment; wherein multiple clients connect to a monitor over a network spanning multiple hops. Please note that we only control the WiFi part of our end-to-end connection in

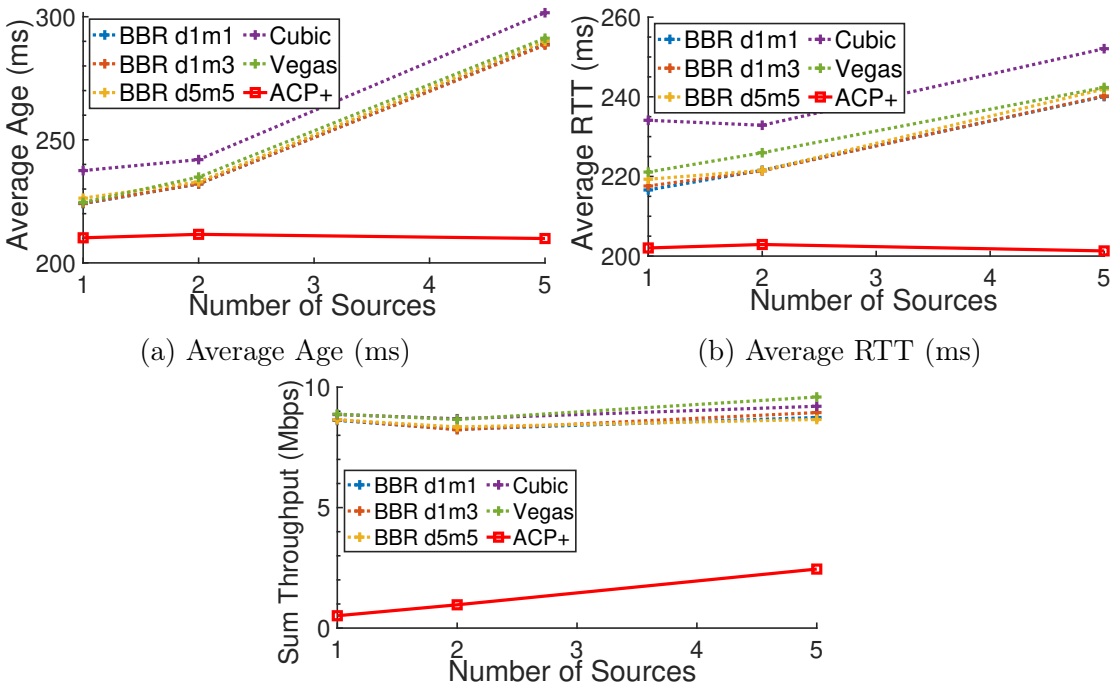
ORBIT, and the rest of the path traverses the public shared Internet. We compare our ACP+ protocol with TCP. We use `iPerf3` [162] to generate TCP traffic from ORBIT nodes towards the AWS server. To observe the impact of contention on the wireless last-mile, we experiment with the number of connected clients to the WiFi access points in two different configurations. The *low contention* configuration includes five or fewer source nodes connected to the same WiFi access point, which closely emulates smart home-like scenarios. On the other hand, experiments with more than five (and up to 80) WiFi nodes are classified as *high contention*. They are representative of smart factory scenarios.

We perform at least five runs of each experiment in each configuration setting and present averages of performance metrics of interest. Our experiments lasted several months and were repeated across different days of the week and at different times of the day. We detail our network configurations and the results obtained in the following sections.

4.6.1 Shared Network with Low Contention

The low WiFi contention configuration mimics the IoT smart home scenario where there are typically numbered devices sharing the network. We compare ACP+ with the TCP congestion control algorithms CUBIC [146], Vegas [147] and BBR [32].

Note that all three algorithms are popularly used on the Internet and are the commonly chosen examples of loss-based, delay-based, and hybrid congestion control, respectively (see Section 4.2). Figure 4.8 compares the different TCP congestion control algorithms and ACP+ in a low WiFi access contention environment. The number of nodes connected to the access point are a maximum of 5. The WiFi link rate is set to 12 Mbps. Figure 4.8a shows the average age achieved by the TCP control algorithms and ACP+. ACP+ performs better than all the chosen congestion control algorithms and the gap between ACP+ and the rest increases as the number of clients increase from 1 to 5. In Figure 4.8b, we see a rise in the average delay (RTT) that each TCP segment sees as the number of nodes increases. This explains the above-stated increase in age. ACP+, on the other hand, achieves a similar average age and RTT per client. The TCP algorithms,



(a) Average Age (ms)

(b) Average RTT (ms)

(c) Sum Throughput (Mbps)

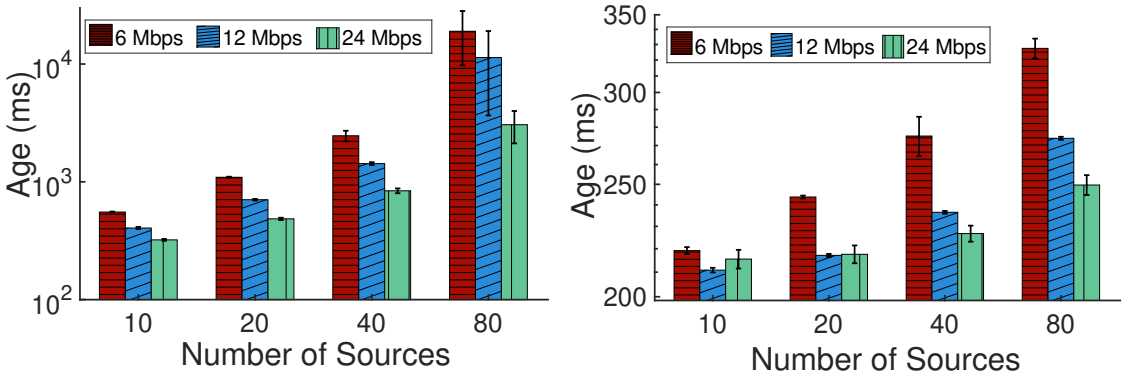
Figure 4.8: Comparison of different TCP Congestion Control Algorithms and ACP+ for low WiFi access contention. The WiFi link rate was set to 12 Mbps.

unlike ACP+, always try and fully utilize the bottleneck WiFi link. As is seen in Figure 4.8c, TCP always achieves a sum throughput of about $8-9$ Mbps, which is close to saturating the 12 Mbps WiFi link when we also include packet header overheads. TCP's mechanism of filling the network pipe leads to queue buildup in the network, leading to an increase in delays.

Recall our results from Section 3.12 for ACP+. The per source throughput at which age is minimized is very low. For a small enough number of sources, the different ACP flows are oblivious to each other. The sum throughput, shown in Figure 4.8c, is far from saturating the 12 Mbps WiFi link.

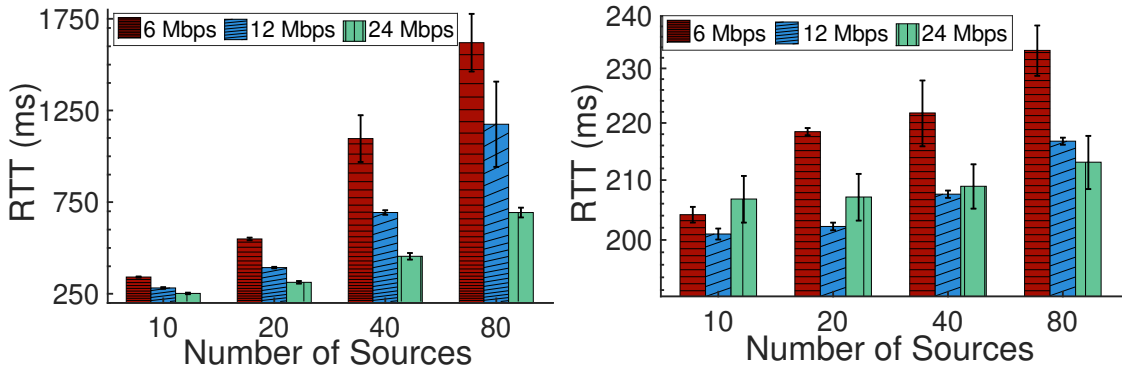
4.6.2 Shared Network with High Contention

In this network configuration, we only compare ACP+ and BBR since BBR outperforms CUBIC and Vegas in the core network (Section 4.4) and also when WiFi access contention is low (Section 4.6.1). We elucidate the impact of the bottleneck bandwidth on BBR by setting the WiFi link rates to 6, 12, and 24 Mbps, using



(a) TCP BBR (b) ACP+

Figure 4.9: Average Age achieved by TCP BBR and ACP+ in the presence of high WiFi access contention. The solid region indicates the mean, and the bar indicates the standard deviation across different runs. Note the different scales on the y-axes.

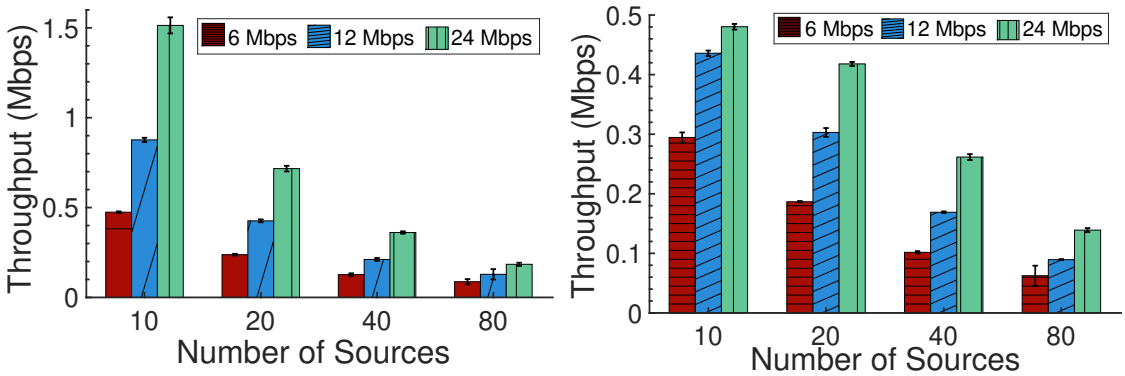


(a) TCP BBR (b) ACP+

Figure 4.10: Average RTT achieved by TCP BBR and ACP+ in the presence of high WiFi access contention.

the `iwconfig` [140] utility.

Figure 4.9 compares the age achieved by BBR and ACP+ for 10, 20, 40 and 80 clients connected to a fixed-rate WiFi access point, for rates 6, 12, and 24 Mbps. Observe in Figure 4.9a that as we increase the number of clients, for a given WiFi link rate, we see a very rapid increase in the average age per node achieved by BBR. While ACP+ also sees an increase in age with the number of clients (sources), the increase is less rapid than in the case of BBR. Also, the age achieved by ACP+ is much smaller than that achieved by BBR, especially for 20 or more clients sharing the access. When using BBR, for a link rate of 24 Mbps, age increases ten fold from 321.5 ms to 3054.7 ms as the number of clients increase from 20 to 80. The corresponding age values for ACP+ are 215.5 ms and 249.65 ms. Given BBR's performance, and the fact that it performs better age-wise than



(a) TCP BBR (b) ACP+

Figure 4.11: Average Throughput achieved by TCP BBR and ACP+ in the presence of high WiFi access contention. The solid region indicates the mean, and the bar indicates the standard deviation across different runs. Note the differences in scale on the y-axes.

the other congestion control algorithms, the TCP algorithms are unsuitable for age control in high access contention settings.

The large age values when using BBR are explained by the corresponding large RTT shown in Figure 4.10a. The RTTs are always larger than the corresponding RTTs when using ACP+, which are shown for comparison in Figure 4.10b. As with age, RTT also increases rapidly with the number of clients when using BBR.

Consider Figures 4.11a and 4.11b that respectively show BBR and ACP+ throughputs. While BBR has larger throughputs than ACP+, the throughputs for when there are 20 or more clients are similar. For when there are 10 clients, ACP+ has much smaller throughputs for WiFi link rates of 12 and 24 Mbps. This is because the ACP+ paths do not together fully utilize the WiFi link for the rates, since the constraining factor as regards optimization of age is the backhaul beyond the access (see Section 3.12). For 10 clients at 6 Mbps and for when there are 20 or more clients, ACP+ and BBR have similar throughputs. However, BBR achieves the throughputs at a much larger RTT and hence achieves a much larger age. The large RTTs when using BBR are likely because all BBR clients attempt to fully utilize the bottleneck link, which is the WiFi link in our experiments. ACP+, on the other hand, keeps the backlog in the end-to-end path small.

We end with the WiFi packet retry rates that we obtained by configuring a node in the ORBIT testbed as a sniffer. One would expect larger client densities to witness higher rates of retries due to a higher rate of packet decoding errors that

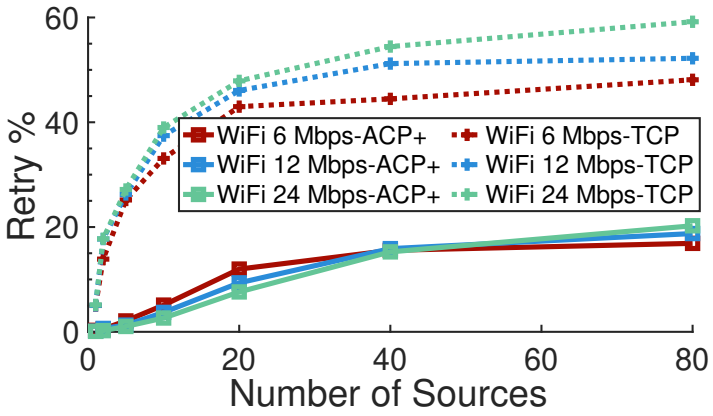


Figure 4.12: WiFi retry rates for TCP and ACP+ for different network configurations.

result from collisions over the shared WiFi medium access. Figure 4.12 shows the percentage of WiFi retries observed when using BBR and ACP+. As we increase the number of clients, the retry rates when using BBR approach 50%. The retry rates for ACP+ saturate to a much lower $\approx$ 20%. The significantly larger retry rates when using BBR corroborate the higher RTT, both of which are explained by TCP clients attempting to saturate the bottleneck link. Further note that for TCP BBR, the higher the WiFi link rate, the higher is the retry percentage. This is because the congestion control algorithm estimates the bottleneck link rate and pushes a larger amount of packets for higher link rates.

4.7 Chapter Summary

The chapter details an in-depth real-world study on ageing when using a mix of loss-based, delay-based and hybrid TCP congestion control algorithms over an end-to-end Internet path. We considered paths only over the core network and also those with a shared access as the first hop. Comparisons with ACP+ demonstrate that the congestion control algorithms are always worse age-wise. In fact, when 20 or more nodes share an access, the algorithms are unsuitable for age control over the Internet.

Chapter 5

Coexistence of Age Sensitive Traffic and High Throughput Flows: Does Prioritization Help?

5.1 Introduction

As discussed in previous chapters, IoT devices often communicate their updates, which require timely delivery to a server in the cloud, over an end-to-end path that includes a shared wireless access followed by a multihop path over the Internet to the server. The update traffic often shares the path with traffic that would like to achieve high throughput. Update packets that require timeliness will suffer large delays if queued together with high throughput flows. They may also suffer significant delays in obtaining transmission opportunities over a shared multiaccess when competing for the same with high throughput flows. In practice, the networking stack allows priorities to be associated with data flows using, for example, the mechanism of Differentiated Services Code Point (DSCP). In principle, this can help alleviate the adverse consequences of update packets sharing the network with throughput flows.

In this chapter, we empirically shed light on the benefits of prioritizing update packets sent over a shared WiFi access to a server in the cloud [14]. We use the time

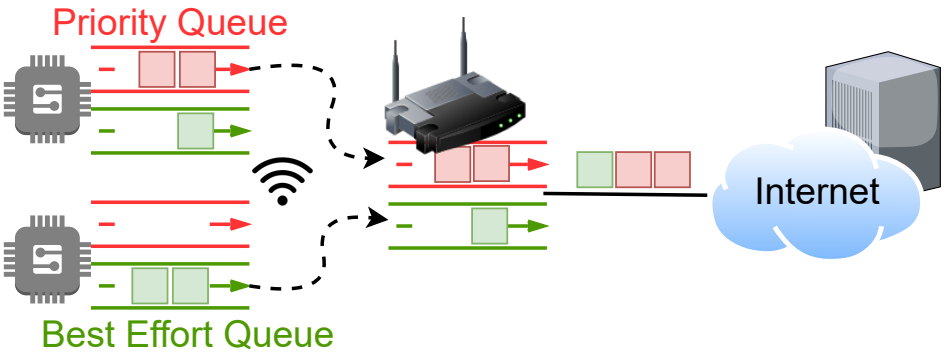


Figure 5.1: Illustration of priority queueing over a shared WiFi access. Nodes and AP maintain separate queues for different service classes.

average age of information (AoI) [2] to quantify timeliness. Transmission Control Protocol (TCP) flows are used to emulate high throughput traffic. For end-to-end flows carrying update packets, we regulate the end-to-end rate of updates using the Age Control Protocol (ACP+), which has been shown to provide good timeliness performance over paths of interest in our previous chapters.

Work on optimizing metrics of the age of information has considered packet management techniques, including priorities and preemption when multiple sources share a service facility [163, 164, 165, 166, 167, 168, 169]. Such work often uses queueing models to capture sharing of the network resources. However, contention has not been modeled when sources share a multiaccess channel. Also, these works assume that all traffic sharing the facility requires timely delivery. Last but not least, it is often assumed that packet management may discard a source packet.

Given the shared WiFi access and Enhanced Distributed Contention Access (EDCA), we have different queues for the ACP+ and TCP flows in our work as we assign a higher priority to update packets (ACP+ flows). The queues, however, are FCFS and don't allow preemption. Our specific contributions are:

1. We provide an empirical evaluation of the impact of coexisting ACP+ and TCP flows on the time-average age of information of the ACP+ flows and the throughputs of the TCP flows. Both flows share a WiFi network and have a server in the cloud as their destination.

2. Using different experimental configurations (a) with and without prioritization, (b) with and without shared access, and (c) in the absence of TCP flows, we show that while giving ACP+ flows higher priority in the absence of contention over the WiFi access (all flows originate from the same WiFi client) effectively isolates the ACP+ flows from the TCP flows, the contention that is caused when WiFi access is introduced, and all flows originate from different WiFi clients results in barely any gains from prioritization.

3. We show from our experiments that as the number of ACP+ flows become large enough, TCP and ACP+ flows (prioritized) sharing the same WiFi access is worse both in terms of the throughputs of the TCP flows and the timeliness achieved by the ACP+ flows.

5.2 Related Work

Several works [163, 164, 165, 166] analyze the average age of updates in the presence of priority traffic. In [163], the authors analyze the average age of updates when the sources are assigned different priorities for two service facilities. One which allows source agnostic preemption in service by a new arrival of equal or higher priority and the other in which there is a waiting room of size 1 and a new arrival can preempt an update in waiting but not in service. In [164], the authors expand the waiting room to allow each source to have up to one waiting update while the server is busy. This also allows an update in service that is preempted by a higher priority source to be saved in the waiting room to resume service later. In [165], the authors analyze peak age when sources have priorities and queues are of infinite size for Poisson arrivals and general service times. In [166], the authors propose and analyze three source aware packet management policies considering a memoryless service facility of a single queue and server. The facility sees arrivals from two independent Poisson sources. The policies make different choices regarding the size of the waiting room and whether preemption is allowed in the service. In [168], the update currently in service is preempted instead of discarding a new arrival. In [169] arrivals consist of a mix of ordinary and priority updates. The latter can preempt any update in service. In case the preempted update is ordinary, it is not discarded and is queued for resuming service later.

5.3 Prioritization in Networks

Several mechanisms exist in modern networked systems that commonly address network bottlenecks by allowing priority packets to pass first [170, 171]. The majority of such mechanisms operate by categorizing network traffic into distinct "service classes" – each one assigned a separate queue. Based on the QoS demands of each class, these mechanisms manage the rate of each class queue such that the services can access a bottleneck link depending on their priority. For example, a router at the bottleneck link may handle voice-over-IP (VoIP) application traffic using a high-priority, high-rate queue, while packets of video streaming applications over the same link might be forwarded at a significantly lower rate.

IETF Diffserv Service Class	DSCP	802.11 Access Category	User Priority
Network Control	CS7, CS6	`AC_VO` (Voice)	7
Signaling	CS5	`AC_VI` (Video)	5
Multimedia Conferencing/ Streaming	AF41-43, AF31-33	`AC_VI` (Video)	4
High Throughput Data	AF11-13	`AC_BE` (Best Effort)	3
Low-Priority Data	CS1	`AC_BK` (Background)	1

Table 5.1: Diffserv QoS mapping in wired (DSCP) and WiFi access.

There are several ways in which network operators can classify network flows into different service classes in their managed routers. For example, operators may use the destination IP address and port to identify an application type (e.g., data egress from Netflix servers) or prioritize based on the transport protocol used (RTP may have a different priority than UDP/TCP traffic) [172]. The most common traffic classification method uses Differentiated Services Code Point (DSCP) markings. Application providers can assign their packets with a unique code in the IP layer. Each code maps to a unique traffic class type that can be treated with a different priority. The current standard dictates network management control traffic to be assigned the highest priority, followed by interactive applications, low-loss low-latency data transfers, and finally, best-effort data transfer applications [170, 171]. As the DSCP value is embedded in the IP header (layer 3) of every packet, it is visible to every router on the Internet and thus allows for end-to-end flow prioritization.

However, since multiaccess schemes like WiFi operate at layer 2 i.e., MAC (medium access control) of the networking stack, they remain oblivious to DSCP markings in the IP layer. Instead, the 802.11 standard employs its prioritization using Enhanced Distributed Channel Access (EDCA) or Hybrid Controlled Channel Access (HCCA) [173]. Similar to DSCP, the 802.11 prioritization assigns eight separate queues at the MAC layer in which data packets are segregated based on

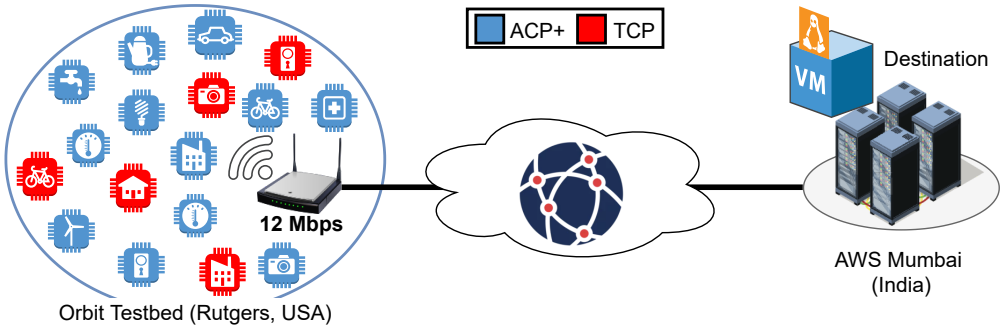


Figure 5.2: An illustration of the network topology. Clients containing a mix of TCP (red) and ACP+ (blue) are connected to a WiFi AP located in the Orbit Testbed's WiFi grid in USA. The server is located in AWS Mumbai, India.

their priority level (defined as `User Priority`). Each priority level is assigned to one of the four access categories (analogous to DSCP traffic classes), i.e. background (`AC_BK`), best-effort (`AC_BE`), video (`AC_VI`) and voice (`AC_VO`) (arranged in increasing priority order). Each access category uses different CSMA/CA minimum and maximum contention window sizes and also inter-frame spacing (IFS). This enables packets belonging to a higher priority access category faster access to the shared channel and less contention from lower priority packets awaiting access.

Recent efforts have mapped DSCP markings to 802.11 EDCA priority and access categories [174]. It is now possible for application providers to assign their traffic higher priority in both wired and wireless networks by setting DSCP in the IP header. Table 5.1 summarizes different traffic classes and their priority mappings between DSCP (Diffserv) and 802.11.

5.4 Experimental Setup and Methodology

Figure 5.2 illustrates our real-world experimental setup. Our setup and methodology is similar to our previous experiments discussed in Section 3.12 of Chapter 3. For our experiments, we use the ORBIT testbed [138], which is an open wireless network emulator grid located in Rutgers University, USA. ORBIT houses multiple programmable radio nodes deployed in a controlled grid-like environment with adjacent WiFi nodes along rows and columns at a distance of 1 m from each other. Each ORBIT node runs Ubuntu 18.04 LTS over Linux kernel v5.0. By default,

ORBIT nodes use the 1 Gbps ethernet NIC to connect to the Internet. We set up one of the ORBIT nodes as an 802.11n access point configured to operate at 5 GHz on a fixed channel using `hostapd` and the Atheros (`ath9k`) Linux WiFi driver [139]. To focus on the interplay between priorities and contention, we disable the automated WiFi physical layer (PHY) rate control in `ath9k` drivers and instead use a *fixed* WiFi PHY rate for the length of an experiment. While most of our experiments use a PHY rate of 12 Mbps, we also use 6 Mbps for some experiments. We provide experiment specific PHY rates in Section 5.5. We select up to 80 nodes as WiFi clients in the ORBIT testbed and associate them to the ORBIT node configured as the WiFi access point. Our WiFi access point routes every packet received over WiFi to the public Internet over Ethernet. We also set up a node in the testbed as a *sniffer* that captures all packets sent over the WiFi channel during our experiments. The sniffer data allows us to estimate MAC layer packet retry percentages suffered by the ACP+ and TCP flows over the WiFi access. It also helps confirm that EDCA priorities have been applied. We use an ec2 AWS instance in Mumbai, India, as our destination server for all flows. The baseline round-trip-time (RTT_{base}), calculated by sending one packet for every `ACK` between our WiFi clients and the server is $\approx$ 200-210 ms. We evaluate three different flow configurations.

1. `ACP-default`. Update packets are sent over an end-to-end path between a WiFi client (the ACP+ source) and the AWS server (ACP+ monitor) using the Age Control Protocol (ACP+) [29, 31]. Update packets sent by ACP+ are given the default priority and treated as best-effort traffic.

2. `ACP-priority`. It is same as `ACP-default` but here ACP+ packets are given the highest network priority by setting the DSCP value as CS7 (see Table 5.1).

3. `TCP-iperf`. We use `iperf3` to generate TCP traffic from WiFi clients to the AWS server. We configure each TCP flow to use the `cubic` congestion control [146]. TCP flows are always treated as best effort in our work.

In addition to priority queueing at our configured WiFi access (available default in the 802.11 standard), we use `CAKE` [175] at our AP node to support QoS priority

at the Ethernet interface. `CAKE` is a comprehensive network queue management utility that has been deployed as part of the OpenWRT framework and is available in all Linux kernels version 4.19 and later [176]. `CAKE` supports Differentiated Services (DiffServ) prioritization scheme and maps `ACP-priority` flows to the highest priority queue (reserved for voice applications) ingressing the Ethernet interface. On the other hand, `CAKE` treats flows belonging to `ACP-default` and `TCP-iperf` as the lowest priority best-effort traffic.

We use three different experiment configurations and simultaneously run ACP+ and TCP flows to evaluate the gains from prioritizing ACP+ flows. Specifically, in *Baseline Priority* (BP) we initiate one or more `ACP-priority` and `TCP-iperf` flows from a single WiFi client. This setting eliminates any interference between the flows due to contention over the WiFi access. It focuses on the co-existence of ACP+ prioritized flows and TCP flows in what effectively is a setting with a single server and one FCFS queue for every priority class. In *Multiaccess Priority* (MP), each `ACP-priority` and `TCP-iperf` flow runs on a separate WiFi client. The flows therefore compete for the shared WiFi multiaccess, resulting in contention. Lastly, in *Best Effort* (BE), as in MP, each flow begins in a different WiFi client. We have `ACP-default` flows where no priority is assigned along with `TCP-iperf` flows. All flows are thus treated as best effort.

Evaluation Metrics. We now define our evaluation metrics. The performance of an ACP+ flow (`ACP-default` or `ACP-priority`) is evaluated in terms of the estimate of *time-average age* [2] at the source. Note that since the source of the flow (a WiFi client) is not time-synchronized with the AWS server, age can't be calculated at the server. We bank on ACP+ `ACK` packets sent by the server back to the source in response to every update packet sent by the source to estimate the age. The round-trip-time (RTT) corresponding to an `ACK-ed` source packet is assumed to be the packet's system time. Age is assumed to reset to this time when the source receives an `ACK`. *Out-of-sequence* older `ACK`s are discarded, which is in line with the *freshness* requirement. Using RTTs as an estimate of system time can lead to over-estimation of age. However, since we consider a linear age function, the bias in estimation does not affect the optimal operating point. Chapter 3 contains the detailed design principles and operation of ACP+. In section 5.5, we

present the *mean time-average age*, which is the mean calculated over all ACP+ flows.

We also discuss ACP+ throughput, which is the end-to-end rate (Mbps) of `ACK-ed` source packets and is calculated by the source based on the `ACK` packets it receives and the size of sent update packets. An update packet is of size 1024 bytes in all our experiments. We will also present *WiFi MAC packet retry percentages*. These simply capture the percentage of packets on air that were *retries* for a source. The retry packets are marked with a retry flag which is captured by the sniffer. Last but not least, TCP throughput is also a metric of interest. For all metrics, we present the mean calculated over 3 repeats of an experiment, where each experiment is 1000 seconds long.

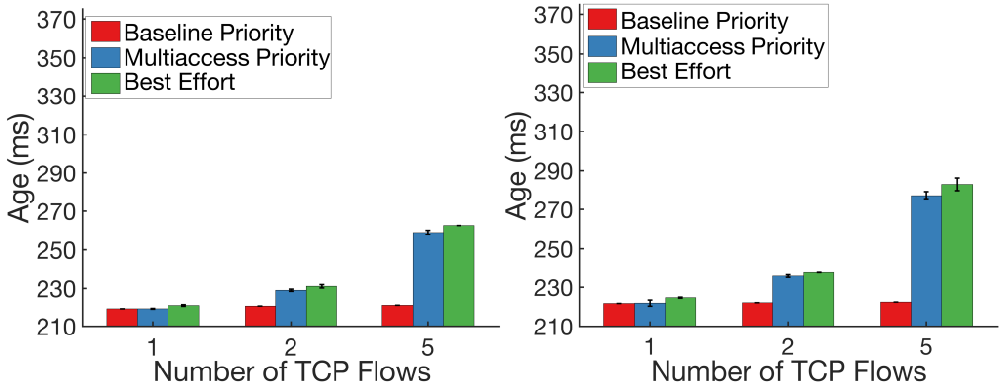
5.5 Evaluation

We discuss our observations from experiments performed using the methodology described in section 5.4. They help gain insight into whether prioritizing benefits ACP+ flows when they share a WiFi access with TCP flows.

5.5.1 Analyzing Gains from Prioritizing ACP+ Flows over the Shared WiFi Multiaccess

Figure 5.3 shows the mean time-average age achieved by ACP+ flows sharing the WiFi network with TCP flows. Figures 5.3a, 5.3b, 5.3c and 5.3d show the mean age for when the number of ACP+ flows are set to 2, 5, 10, and 20 respectively. For each selection of a number of ACP+ flows, the mean age is shown for when the number of TCP flows are 1, 2, and 5. Further, for each selection of number of ACP+ and TCP flows, the mean is shown for the network configurations of *Baseline Priority* (BP), *Multiaccess Priority* (MP) and *Best Effort* (BE).

Contention over the shared WiFi multiaccess increases in the configurations MP and BE when the number of ACP+ clients or TCP clients increases. Let's begin by considering the mean age achieved under *Baseline Priority*. For a given number of ACP+ flows, for example, 5 flows in Figure 5.3b, the mean age stays



(a) Two ACP+ flows

(b) Five ACP+ flows

(c) Ten ACP+ flows

(d) Twenty ACP+ flows

Figure 5.3: Mean time-average age achieved by 2, 5, 10 and 20 ACP+ flows for *Baseline Priority*, *Multiaccess Priority* and *Best Effort* configurations for 1, 2, and 5 coexisting TCP flows.

unchanged for different numbers of TCP flows. For 5 ACP+ flows, this age is $\approx$ 222 ms for 1, 2, and 5 TCP flows. The age is $\approx$ 230.5 ms when there are 20 ACP+ nodes as in Figure 5.3d. The age stays the same for a given number of ACP+ nodes because in BP all ACP+ and TCP flows originate in the same WiFi client. Because of their higher priority than TCP flows, ACP+ flows are unaffected by changes in the number of TCP flows originating in the WiFi client. On the other hand, an increase in the number of ACP+ flows does result in an increase in the mean age. This is because updates from a larger number of ACP+ flows share the same priority queue in the WiFi client. Mean age increases from $\approx$ 220 ms for 1 - 2 ACP+ flows (Figure 5.3a) to $\approx$ 230 ms for 20 ACP+ flows (Figure 5.3d).

As seen in Figure 5.3 for MP and BE configurations in which flows are distributed over different WiFi clients sharing the WiFi multiaccess, mean age in-

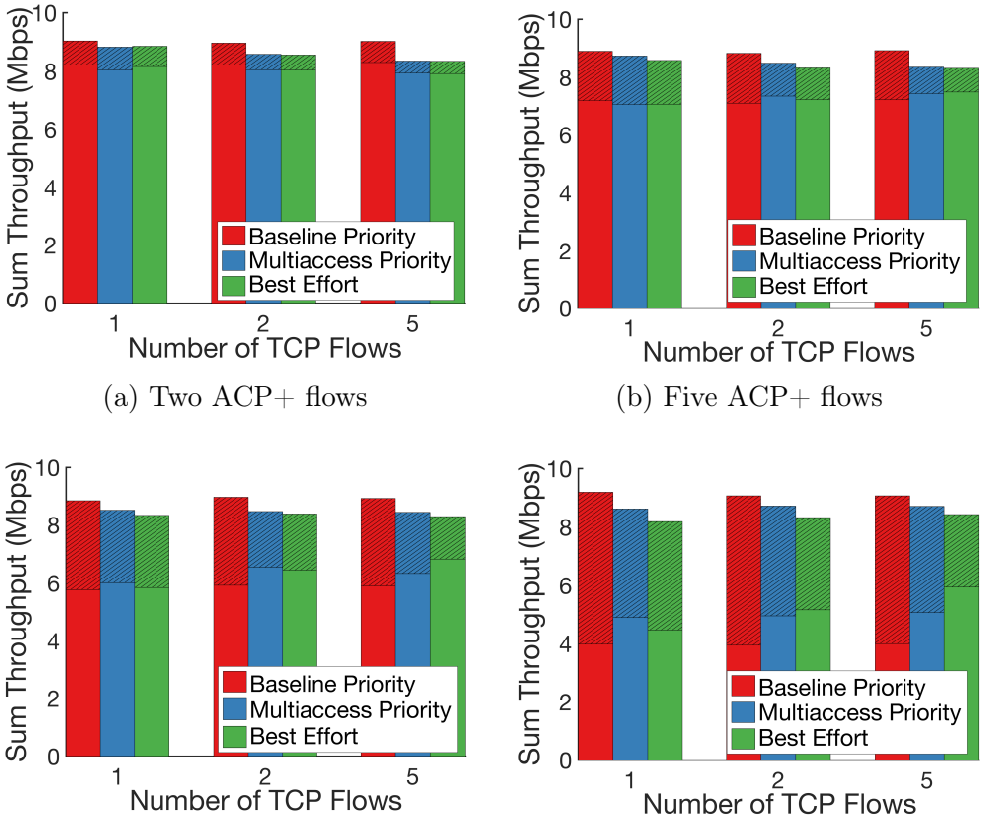
creases significantly as the number of TCP flows increases for any selection of the number of ACP+ flows. Assigning a higher priority to ACP+ flows, as in MP, is ineffective in isolating them from the effects of TCP flows sharing the multiaccess. Also, the mean age when using *Multiaccess Priority* is in general not much smaller than when treating ACP+ with the same priority as TCP when using *Best Effort*.

To understand the reason behind significantly worse mean age when using MP, we begin by considering the throughput obtained by the TCP flows. Later we also look at the WiFi MAC layer retry percentages suffered by update packets of ACP+ flows and also their round-trip times (RTTs).

Further, we observe from Figure 5.3 that for any selected number of TCP and ACP+ flows, contention over the multiaccess results in a higher mean age in comparison to BP. In fact, even for just 2 ACP+ and 2 TCP flows, we see that the mean age for the setting of MP is about 9 ms more than that for *Baseline Priority*. This gap increases rapidly with an increase in the number of ACP+ and TCP flows. For example, it jumps to 38 ms for 2 ACP+ and 5 TCP flows, is 55 ms for when we have 5 ACP+ and 5 TCP flows, and is 130 ms for 20 ACP+ and 5 TCP flows.

Figure 5.4 shows the sum throughput (sum of throughputs of all ACP+ and TCP flows) for the network configurations BP, MP, and BE, and different numbers of TCP and ACP+ flows. It can be observed that the sum throughput is about the same for all the configurations and all numbers of TCP and ACP+ flows. It stays in the very narrow range of 8.8 to 9 Mbps. Essentially, the TCP and ACP+ flows together achieve the available data payload rate of about 9 Mbps, given the link rate of 12 Mbps. The figure also shows the share of ACP+ flows and that of TCP flows in the sum throughput. As can be seen, the fraction of sum throughput that corresponds to ACP+ flows increases with the number of ACP+ flows. As expected, the sum throughput of ACP+ flows for *Baseline Priority* is only a function of the number of ACP+ flows and is not impacted by the number of TCP flows. This throughput is 0.8 Mbps for when we have 2 ACP+ flows and goes up to about 5 Mbps for 20 ACP+ flows.

Further note, for a given number of ACP+ flows, the sum throughput of TCP flows stays about the same for the configurations BP, MP, and BE. For BE, it is



(a) Two ACP+ flows

(b) Five ACP+ flows

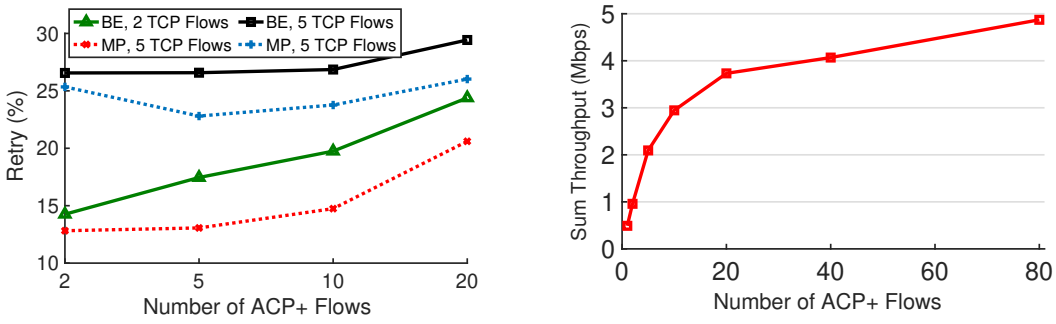
(c) Ten ACP+ flows

(d) Twenty ACP+ flows

Figure 5.4: Sum throughput of ACP+ and TCP flows with their respective shares for 2, 5, 10 and 20 ACP+ flows. For each stacked bar, the diagonally striped top part corresponds to the sum of ACP+ flows and the bottom part shows the sum TCP throughput. For each number of ACP+ flows, the throughputs are shown for 1, 2, and 5 coexisting TCP flows and for *Baseline Priority*, *Multiaccess Priority* and *Best Effort*.

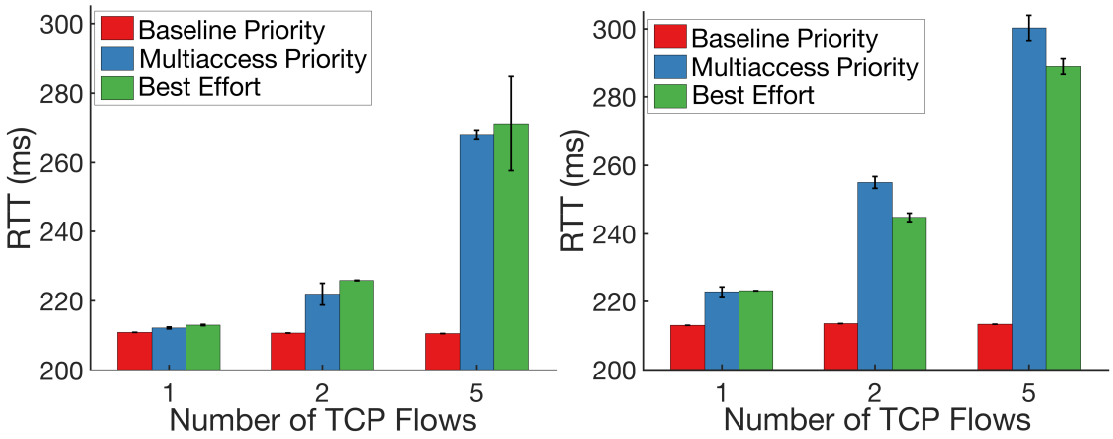
within ≈ 2 Mbps of sum TCP throughputs for *Baseline Priority*. Specifically, for larger numbers of ACP+ flows, the TCP sum throughput is greater by at most ≈ 1 Mbps when using *Multiaccess Priority* compared to using BP. When using BE, it is at most ≈ 2 Mbps higher. TCP throughput benefits in BE because ACP+ flows have the same access priority as TCP flows. The above observation tells us that the significant increases in mean age seen in Figure 5.3 with an increase in the number of TCP flows for a given number of ACP+ flows, for MP and BE, may not be entirely attributed to TCP's throughput share.

While an increase in TCP flows doesn't impact the TCP sum throughput, it results in increased MAC layer retries of packets of ACP+ flows. It also results in ACP+ flows experiencing large RTTs. Figure 5.5a shows the packet retry percentages as a function of the number of ACP+ flows for two and five TCP flows



(a) (b)

Figure 5.5: (a) Retry percentages of update packets sent in ACP+ flows as a function of number of ACP+ sources. Percentages are shown for *Best Effort* and *Multiaccess Priority*, and for 2 and 5 TCP flows. (b) ACP+ sum throughput in the absence of TCP as a function of the number of ACP+ flows sharing a 6 Mbps WiFi link.



(a) Ten ACP+ flows (b) Twenty ACP+ flows

Figure 5.6: Mean RTT of ACP+ flows. For ten and twenty ACP+ flows, RTT is shown for *Baseline Priority*, *Multiaccess Priority* and *Best Effort*, and for 1, 2, and 5 coexisting TCP flows.

and the configurations of *Multiaccess Priority* and *Best Effort*. Retry percentages increase by about 5% - 10% when the numbers of TCP flows increase from 2 to 5. We also see higher retry percentages when there are larger numbers of ACP+ flows. Also, observe that for a given number of TCP flows, *Best Effort* sees higher retry percentages than MP. This is because having priority has ACP+ flows see a little less contention over the WiFi multiaccess.

We also look at the mean RTTs of updates packets to see the impact of increased MAC layer retries. Figures 5.6a and 5.6b show, respectively for 10 and 20 ACP+ flows, significant increases in RTT as the number of TCP flows increase

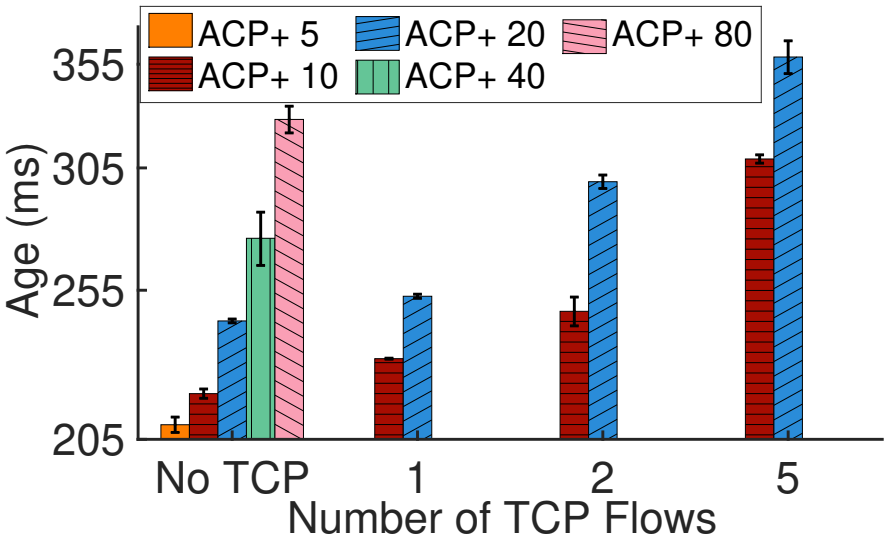


Figure 5.7: Mean time-average age for ACP+ flows with increasing TCP flows. In *No TCP*, all ACP+ flows share a 6 Mbps WiFi link. For all other settings, ACP+ and TCP flows share a 12 Mbps link in *Multiaccess Priority* configuration.

from 1 to 5, for MP and BE. These, together with the retry rates, explain the large mean ages observed when using multiaccess in comparison to when using *Baseline Priority*.

To summarize, the gains from prioritizing ACP+ flows vanish quickly with an increase in contention over the shared WiFi multiaccess. The increased contention leads to higher retries and higher RTTs, resulting in higher time-average age.

5.5.2 Analyzing the Effect of Competing TCP flows on ACP+

In this section, we try to analyze the performance of ACP+ flows sharing a 6 Mbps WiFi link without interference from TCP flows and compare it to the case where all flows are sharing a 12 Mbps WiFi link.

Figure 5.7 shows the mean age achieved by ACP+ flows when they share the WiFi multiaccess of rate 6 Mbps in the absence of TCP flows (labeled *No TCP*) and when the ACP+ and TCP flows share a 12 Mbps in MP configuration. For the *No TCP* setting, we see that the mean age is 209.43, 219.18, 243.79, 275.15, 327.41 ms, respectively for 5, 10, 20, 40, and 80 ACP+ flows. The corresponding sum ACP+ throughputs (see Figure 5.5b) are 2, 2.9, 3.7, 4 and 5 Mbps. So with

80 ACP+ flows sharing the WiFi multiaccess with a link rate of 6 Mbps, and utilizing almost all of it (a sum throughput of 5 Mbps), the mean age is 327.41 ms. Compare these mean ages for the *No TCP* setting with 10 and 20 ACP+ flows under *Multiaccess Priority* when a 12 Mbps WiFi link is shared with 1 - 5 TCP flows (see Figure 5.7). For 10 ACP+ nodes it is 230.7, 247.25, and 309 ms. For 20 ACP+ it is 252.76, 298.9, and 358.77 ms, respectively. Clearly, ACP+ achieves lower age even with 80 flows and lower link rate (of 6 Mbps) when compared with 20 ACP+ flows coexisting with TCP flows even at a higher link rate of 12 Mbps.

For TCP, throughput is the utility of interest. For TCP flows sharing a 6 Mbps WiFi link (without any ACP+ flows), the sum TCP throughput is $\approx$ 5.5 Mbps, which is the expected payload rate after accounting for overheads like packet headers. For ACP+ and TCP flows sharing a 12 Mbps link, the sum throughput is 5 Mbps (1, 2 or 5 TCP flows) for when we have 20 ACP+ flows and is in the range of 6 - 6.5 Mbps for 10 ACP+ flows (see Figure 5.4). In fact, it is only when we have very few ACP+ flows, that the TCP sum throughputs are much larger than 5.5 Mbps. For when TCP shares with only 1 ACP+ flow, the sum TCP throughput is as high as 8 Mbps. With 5 ACP+ flows, the sum throughput ranges from 7 - 7.5 Mbps.

Additionally, the retry percentages for *No TCP* are 2% for 5 ACP+ flows and increase to 17% for 80 ACP+ flows (plot not included). Contrast these with the much higher retry rates in Figure 5.5a for when ACP+ flows share a 12 Mbps link with TCP flows. We also look at the impact of retry rate on RTTs and find that even the RTTs are smaller, with 20 ACP+ flows seeing an RTT less than 220 ms.

To sum up, we found that when 20 ACP+ flows share a 12 Mbps access with TCP flows, having TCP and ACP+ flows use non-interfering 6 Mbps WiFi links is beneficial to both. ACP+ mean ages are much smaller and TCP gets a higher sum throughput of 5.5 Mbps.

5.6 Chapter Summary

We studied the impact of prioritization on the performance of age-sensitive traffic in the presence of competing network traffic. We considered an array of experimen-

tal configurations in real-world network settings. Our results indicate that ACP+ flows gain from prioritization only when contention over the wireless access from competing traffic is low. The gains are non-existent as the contention increases. We also find that a large number of ACP+ and TCP flows using non-interfering 6 Mbps WiFi links results in both better throughput and age performance, respectively for TCP and ACP+ flows, than when the flows share a 12 Mbps access.

Chapter 6

Discussion and Future Research Directions

In this thesis, we provided transport layer solutions for two categories of emerging applications. The first category of applications requires high end-to-end throughput, for example, video streaming and large file downloads. The other category, which includes IoT applications like environmental monitoring and vehicular networks, requires freshness of information at the destination.

For applications that require high end-to-end throughput, we proposed *QAware* [23], which is a novel MPTCP scheduler that enables simultaneous use of multiple available paths. QAware uses end-to-end subflow RTT(s) together with queue occupancies at the layer 2 of the TCP/IP stack to decide the path over which a TCP segment must be sent. Our scheduler outperformed various state-of-the-art schedulers, including DAPS, BLEST, and ECF. QAware's implementation as a modular scheduler for MPTCP Linux kernel version 0.92 has been open-sourced [24].

For applications that desire freshness of information at a destination (monitor), we proposed the *Age Control Protocol* (ACP) [27, 28, 29], which is the first transport layer protocol that enables freshness of information at a destination, wherein the information is sent by a source in the form of status update packets to the destination over the Internet. We evaluated ACP and its improved version ACP+ over a wide range of simulated networks and real-world, end-to-end paths over the Internet. Our evaluation brings insights into age control over the Internet. We also compared and contrasted ACP+ with various state-of-the-art TCP congestion control algorithms. We conclude that the existing algorithms are unsuitable for age control over the Internet.

We end the thesis with an investigation of the coexistence of ACP+ and TCP [14]. We show that the typically proposed prioritization mechanisms do not work in the presence of access contention. We believe that there is a need to model the benefits of prioritization in the presence of contention.

We make ACP+ publicly available at [31].

6.1 Future Research Directions

As discussed in Chapter 2, the QoS requirements of next-generation applications have dictated the need for several improvements and advancements in transport protocols to provide high throughputs and low delays. While this thesis focuses on multipath TCP and proposes a novel cross-layer approach, recent advances in QUIC have opened several new doors within this problem space. Notably, we find the current discussions on multipath QUIC (MP-QUIC) [177] within IETF as an interesting potential extension opportunity for enhancing QAware operation. Specifically, unlike MPTCP, MP-QUIC scheduler design allows exploring a new dimension for multipathing. That is, in addition to scheduling packets over multiple paths for increased reliability, MP-QUIC enables scheduling packets across multiple streams on the same path for increased utilization. As a result, QAware would need to optimize the queueing delay caused by packets scheduled by the application across multiple streams on the same path over every available path to maximize utilization of all available subflows and sub-streams optimally.

Our work in Chapter 3 has highlighted the need for network models that can help us understand age optimization in multi-hop networks that mimic the Internet. Insights obtained from *stop and wait* based protocols are not suitable for age control over the Internet since these protocols don't exploit the presence of multiple hops. Moreover, we need better strategies to enable the coexistence of age-sensitive traffic and high throughput traffic in the presence of access contention, as more than just using prioritization may be required in such settings. Approaches such as time-sensitive networking (TSN) [178] and deterministic networking (DetNet) [179], that provide time-critical networking through the link and network layer advancements show promise if used in conjunction with ACP. We believe that ACP can benefit from such lower layer enhancements, and we plan to explore the interplay of ACP in TSN/DetNet enabled networks in future.

While in this thesis, we focused on applications that either care for *freshness* of data or the throughput, there are emerging Internet applications such as AR, VR,

Autonomous vehicles, teleoperated vehicles, and cloud gaming [180] that care for both the freshness of data along with throughput and reliability. An end-to-end protocol that is able to achieve throughput-age tradeoffs as desired by an application is a possible future research direction. A natural extension, given the use of multiple paths by TCP, is an age control protocol that is able to exploit multiple simultaneous paths. For example, a Multipath ACP. In this case, an Machine Learning based scheduling approach may prove beneficial since it would not only maximize the throughput of an application but also make holistic decisions to minimize end-to-end age across many available paths.

Several additions can be made to the ACP operation to make it applicable to a wider class of next-generation applications. For example, adding packet reliability within ACP may prove useful for remote-control applications (such as robotic control) since control commands require strong transmission guarantees along with low age. Similarly, we also plan to run extensive performance measurements of ACP over more recently available last-mile access technologies, e.g., 5G new radio (NR), LEO satellite links, etc. These technologies use a new spectrum for operation and observe contention, excessive queueing, and interference from the environment much more than traditional LTE/WiFi. Many recent studies have revealed performance bottlenecks of existing transport protocols over these technologies [181, 182, 183]. Based on our learnings from designing QAware, we believe there is a potential for improving ACP performance by considering lower-layer state information, such as driver queue size, SNR, retry/send rate, etc. As such, our focus would be to understand how ACP fares in minimizing end-to-end age over such access technologies and how the control algorithm can be improved to leverage such high bandwidth links effectively.

Last but not least, to have applications and the networking stack adopt age control protocol, we need to work toward standardization. To gain an understanding of recent transport layer innovations, we surveyed standardization efforts being made for new (or extended) transport protocols in IETF [26]. Specifically, we focused on understanding the open problems currently being looked at in the standardization body. We provided valuable insights to researchers in the field on how to design protocols that can later be standardized and used beyond their research study. To this end, we are preparing an IETF standardization draft of ACP [29].

Bibliography

[1] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, *Fundamentals of Queueing Theory*. John Wiley & Sons, 2018, vol. 399.

[2] S. Kaul, R. Yates, and M. Gruteser, "Real-Time Status: How Often Should One Update?" in *Proc. IEEE INFOCOM Mini Conference*, 2012.

[3] "Japan Researchers Break Another Data Speed Record With 1.02 Petabits Per Second Fiber Optic Transfer," https://www.techspot.com/news/94809-japan-researchers-break-another-data-speed-102-petabits.html.

[4] Q. D. Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "Observing Real Smartphone Applications Over Multipath TCP," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 88–93, March 2016.

[5] IoT-Analytics, "Global IoT market size by 2027," "https://iot-analytics.com/iot-market-size", 2021.

[6] M. Liyanage, P. Porambage, and A. Y. Ding, "Five Driving Forces of Multi-Access Edge Computing," *arXiv preprint arXiv:1810.00827*, 2018.

[7] A. T. Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash, "Adaptive 360-degree Video Streaming Using Scalable Video Coding," in *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 2017, pp. 1689–1697.

[8] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, "Low-Latency Networking: Where Latency Lurks and How to Tame It," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 280–306, 2018.

[9] D. M. West, "How 5G Technology Enables the Health Internet of Things," *Brookings Center for Technology Innovation*, vol. 3, pp. 1–20, 2016.

[10] "Bandwidth And Latency Requirements For Generic Applications," "https://epthinktank.eu/2016/01/05/5g-network-technology-putting-europe-at-the-leading-edge/bandwidth_and_latency_requirements/", 2019.

[11] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform For High-Performance Internet Applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[12] A. Begen, T. Akgul, and M. Baugher, "Watching Video Over the Web: Part 1: Streaming Protocols," *IEEE Internet Computing*, vol. 15, no. 2, pp. 54–63, 2010.

[13] Daniel Wagner, "Motion to Photon Latency in Mobile AR and VR," https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c48092, 2018.

[14] T. Shreedhar, S. K. Kaul, and R. D. Yates, "Coexistence of Age Sensitive Traffic and High Throughput Flows: Does Prioritization Help?" in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–6. [Online]. Available: 10.1109/INFOCOMWKSHPS54753.2022.9798247

[15] "Transmission Control Protocol (TCP)," RFC 793, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt

[16] "User Datagram Protocol (UDP)," RFC 768, 1980. [Online]. Available: https://www.ietf.org/rfc/rfc768.txt

[17] M. Handley, "Why the Internet Only Just Works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, 2006.

[18] IBM, "Video Streaming Recommended Settings," "https://support.video.ibm.com/hc/en-us/articles/207852117-Internet-connection-and-recommended-encoding-settings", 2019.

[19] Internet Engineering Task Force (IETF), "Architectural Guidelines for Multipath TCP Development," https://tools.ietf.org/html/rfc6182, 2011.

[20] O. Bonaventure, M. Handley, and C. Raiciu, "An Overview of Multipath TCP," *; login:*, 2012.

[21] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental Evaluation of Multipath TCP Schedulers," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, ser. CSWS '14, 2014.

[22] N. Mohan, T. Shreedhar, A. Zavodovski, J. Kangasharju, and S. K. Kaul, "Is Two Greater Than One?: Analyzing Multipath TCP over Dual-LTE in the Wild," 2019. [Online]. Available: arxiv.org/abs/1909.02601

[23] T. Shreedhar, N. Mohan, S. K. Kaul, and J. Kangasharju, "QAware: A Cross-Layer Approach to MPTCP Scheduling," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 2018, pp. 1–9. [Online]. Available: 10.23919/IFIPNetworking.2018.8696843

[24] T. Shreedhar, N. Mohan, S. K. Kaul, and J. Kangasharju. (2018) **[Code]** MPTCP QAware - GitHub. [Online]. Available: github.com/tanyashreedhar/mptcp-QueueAware

[25] T. Shreedhar, R. Panda, S. Podanev, and V. Bajpai, "Evaluating QUIC Performance over Web, Cloud Storage and Video Workloads," *IEEE Transactions on Network and Service Management*, pp. 1–16, 2021. [Online]. Available: 10.1109/TNSM.2021.3134562

[26] M. Kosek, T. Shreedhar, and V. Bajpai, "Beyond QUIC v1: A First Look at Recent Transport Layer IETF Standardization Efforts," *IEEE Communications Magazine*, vol. 59, no. 4, pp. 24–29, 2021. [Online]. Available: 10.1109/MCOM.001.2000877

[27] T. Shreedhar, S. K. Kaul, and R. D. Yates, "(Poster) ACP: Age Control Protocol for Minimizing Age of Information over the Internet," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 699–701. [Online]. Available: 10.1145/3241539.3267740

[28] T. Shreedhar, S. K. Kaul, and R. D. Yates, "An Age Control Transport Protocol for Delivering Fresh Updates in the Internet-of-Things," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2019, pp. 1–7. [Online]. Available: 10.1109/WoWMoM.2019.8793011

[29] T. Shreedhar, S. K. Kaul, and R. D. Yates, "An Empirical Study of Ageing in the Cloud," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–6. [Online]. Available: 10.1109/INFOCOMWKSHPS51825.2021.9484567

[30] T. Shreedhar, S. K. Kaul, and R. D. Yates, "ACP+: An Age Control Protocol for the Internet," 2022. [Online]. Available: https://arxiv.org/abs/2210.12539

[31] T. Shreedhar, S. K. Kaul, and R. D. Yates. (2021) **[Code]** Age Control Protocol - GitHub. [Online]. Available: github.com/tanyashreedhar/AgeControlProtocolPlus

[32] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[33] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "QTCP: Adaptive Congestion Control with Reinforcement Learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2019.

[34] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "End-to-End Congestion Control Approaches for High Throughput and Low Delay in 4G/5G Cellular Networks," *Computer Networks*, vol. 186, p. 107692, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620312974

[35] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011.

[36] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic Mobility with Multipath TCP," in *Proceedings of the Sixth International Workshop on MobiArch*, ser. MobiArch '11, 2011.

[37] T. Shreedhar, D. Zeynali, O. Gasser, N. Mohan, and J. Ott, "[Under Submission] A Longitudinal View at the Adoption of Multipath TCP," in *IEEE/ACM Transactions on Networking*, 2022, pp. 1–15. [Online]. Available: arxiv.org/abs/2205.12138

[38] F. Aschenbrenner, T. Shreedhar, O. Gasser, N. Mohan, and J. Ott, "From Single Lane to Highways: Analyzing the Adoption of Multipath TCP in the Internet," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–9. [Online]. Available: 10.23919/IFIPNetworking52078.2021.9472785

[39] Apple. (2017) Use Multipath TCP to Create Backup Connections for iOS. [Online]. Available: https://support.apple.com/en-us/HT201373

[40] Android Authority. (2015) South Korea's KT launches 1.17Gbps GiGA LTE. [Online]. Available: https://www.androidauthority.com/kt-launches-1gbps-giga-lte-617147/

[41] Netdev Group. (2020) MPTCP Linux kernel Upstream. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/commit/?id=f870fa0b5768842cb4690c1c11f19f28b731ae6d

[42] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking Estimation-Based MPTCP Scheduler for Heterogeneous Networks," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, 2016.

[43] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating Multipath Transport Through Balanced Subflow Completion," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '17, 2017.

[44] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent Delay-Aware Packet Scheduling for Multipath Transport," in *IEEE International Conference on Communications (ICC)*, 2014.

[45] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths," in *Proceedings of the 13th International Conference of CoNEXT*, 2017.

[46] F. Yang, Q. Wang, and P. D. Amer, "Out-of-Order Transmission for In-Order Arrival Scheduling for Multipath TCP," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014.

[47] S. H. Baidya and R. Prakash, "Improving the Performance of Multipath TCP over heterogeneous paths using slow path adaptation," in *2014 IEEE International Conference on Communications (ICC)*, 2014.

[48] J. Hwang and J. Yoo, "Packet Scheduling for Multipath TCP," in *Seventh International Conference on Ubiquitous and Future Networks*, 2015.

[49] Y. Cao, Q. Liu, G. Luo, and M. Huang, "Receiver-Driven Multipath Data Scheduling Strategy for In-Order Arriving in SCTP-based Heterogeneous Wireless Networks," in *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2015.

[50] D. Ni, K. Xue, P. Hong, H. Zhang, and H. Lu, "OCPS: Offset Compensation based Packet Scheduling Mechanism for Multipath TCP," in *2015 IEEE International Conference on Communications (ICC)*, 2015.

[51] D. Ni, K. Xue, P. Hong, and S. Shen, "Fine-grained Forward Prediction based Dynamic Packet Scheduling Mechanism for Multipath TCP in Lossy Networks," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, 2014.

[52] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-layer Scheduler for Video Streaming over MPTCP," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16, 2016.

[53] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath," in *Proceedings of the 12th International on CoNEXT*, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 129–143. [Online]. Available: http://doi.acm.org/10.1145/2999572.2999606

[54] Y. s. Lim, Y. C. Chen, E. M. Nahum, D. Towsley, and K. W. Lee, "Cross-Layer Path Management in Multi-path Transport Protocol for Mobile Devices," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014.

[55] Z. Rosberg and P. Kermani, "Customer Routing to Different Servers with Complete Information," *Advances in Applied Probability*, vol. 21, 1989.

[56] R. R. Weber, "On the Optimal Assignment of Customers to Parallel Servers," *Journal of Applied Probability*, 1978.

[57] W. Whitt, "Deciding Which Queue to Join: Some Counterexamples," *Oper. Res.*, 1986.

[58] W. Winston, "Optimality of the Shortest Line Discipline," *Journal of Applied Probability*, 1977.

[59] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 73–84. [Online]. Available: https://doi.org/10.1145/2535372.2535384

[60] M. T. IETF, "MPTCP Linux implmentation v0.93," http://multipath-tcp.org/pmwiki.php?n=Main.Release93, 2017.

[61] Internet Engineering Task Force (IETF), "BQL enabled drivers," https://www.bufferbloat.net/projects/bloat/wiki/BQL_enabled_drivers/, 2014.

[62] Bufferbloat community, "The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm," https://tools.ietf.org/id/draft-ietf-aqm-fq-codel-06.html, 2014.

[63] "Global Internet phenomenon," www.sandvine.com/.../global-internet-phenomena-report-latin-america-and-north-america.

[64] S. Lederer, C. Müller, and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12, 2012.

[65] T. Arnold, J. He, W. Jiang, M. Calder, I. Cunha, V. Giotsas, and E. Katz-Bassett, "Cloud Provider Connectivity in the Flat Internet," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 230–246. [Online]. Available: https://doi.org/10.1145/3419394.3423613

[66] C. Kam, S. Kompella, and A. Ephremides, "Age of Information Under Random Updates," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, 2013, pp. 66–70.

[67] C. Kam, S. Kompella, and A. Ephremides, "Effect of Message Transmission Diversity on Status Age," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, June 2014, pp. 2411–2415.

[68] C. Kam, S. Kompella, G. D. Nguyen, and A. Ephremides, "Effect of Message Transmission Path Diversity on Status Age," *IEEE Trans. Info. Theory*, vol. 62, no. 3, pp. 1360–1374, Mar. 2016.

[69] I. Johansson and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia," https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-scream-cc-13, 2017.

[70] S. Holmer and H. Lundin and G. Carlucci and L. De Cicco and S. Mascolo, "A Google Congestion Control Algorithm for Real-Time Communication," https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02, 2016.

[71] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing Age of Information in Vehicular Networks," in *IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, Salt Lake City, Utah, USA, 2011.

[72] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," in *The 20th IEEE Real-Time Systems Symposium, 1999. Proceedings.* IEEE, 1999, pp. 32–43.

[73] A. Karakasidis, P. Vassiliadis, and E. Pitoura, "ETL Queues for Active Data Warehousing," in *Proceedings of the 2nd international workshop on Information quality in information systems*, ser. IQIS '05. Baltimore, Maryland: ACM, 2005, pp. 28–39, ACM ID: 1077509.

[74] H. Yu, L. Breslau, and S. Shenker, "A Scalable Web Cache Consistency Architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 163–174, Aug. 1999, ACM ID: 316219.

[75] J. Cho and H. Garcia-Molina, "Effective page refresh policies for web crawlers," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, pp. 390–426, 2003.

[76] A. Kosta, N. Pappas, and V. Angelakis, "Age of information: A New Concept, Metric, and Tool," *Foundations and Trends in Networking*, 2017.

[77] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of Information: An Introduction and Survey," *IEEE Journal on Selected Areas in Communications*, 2021.

[78] M. Costa, M. Codreanu, and A. Ephremides, "Age of Information With Packet Management," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, June 2014, pp. 1583–1587.

[79] L. Huang and E. Modiano, "Optimizing Age-of-Information in a Multi-class Queueing System," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jun. 2015.

[80] M. Costa, M. Codreanu, and A. Ephremides, "On the Age of Information in Status Update Systems With Packet Management," *IEEE Trans. Info. Theory*, vol. 62, no. 4, pp. 1897–1910, April 2016.

[81] C. Kam, S. Kompella, G. D. Nguyen, J. Wieselthier, and A. Ephremides, "Age of Information With a Packet Deadline," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, 2016, pp. 2564–2568.

[82] K. Chen and L. Huang, "Age-of-Information in the Presence of Error," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, 2016, pp. 2579–2584.

[83] L. Huang and L. P. Qian, "Age of Information for Transmissions over Markov Channels," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–6.

[84] J. P. Champati, H. Al-Zubaidy, and J. Gross, "Statistical Guarantee Optimization for Age of Information for the D/G/1 Queue," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2018, pp. 130–135.

[85] Y. Inoue, H. Masuyama, T. Takine, and T. Tanaka, "A General Formula for the Stationary Distribution of the Age of Information and Its Application to Single-Server Queues," *CoRR*, vol. abs/1804.06139, 2018. [Online]. Available: http://arxiv.org/abs/1804.06139

[86] A. Soysal and S. Ulukus, "Age of Information in G/G/1/1 Systems," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 2022–2027.

[87] A. Soysal and S. Ulukus, "Age of Information in G/G/1/1 Systems: Age Expressions, Bounds, Special Cases, and Optimization," *IEEE Transactions on Information Theory*, vol. 67, no. 11, pp. 7477–7489, 2021.

[88] B. T. Bacinoglu, E. T. Ceran, and E. Uysal-Biyikoglu, "Age of Information Under Energy Replenishment Constraints," in *Proc. Info. Theory and Appl. (ITA) Workshop*, Feb. 2015, la Jolla, CA.

[89] R. Yates, "Lazy is Timely: Status Updates by an Energy Harvesting Source," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, 2015.

[90] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, "Update or Wait: How to Keep Your Data Fresh," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7492–7508, 2017.

[91] S. Nath, J. Wu, and J. Yang, "Optimizing Age-of-Information and Energy Efficiency Tradeoff for Mobile Pushing Notifications," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, July 2017, pp. 1–5.

[92] S. Farazi, A. G. Klein, and D. R. Brown, "Average Age of Information for Status Update Systems With an Energy Harvesting Server," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2018, pp. 112–117.

[93] A. Arafa, J. Yang, and S. Ulukus, "Age-Minimal Online Policies for Energy Harvesting Sensors with Random Battery Recharges," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[94] Z. Chen, N. Pappas, E. Björnson, and E. G. Larsson, "Age of Information in a Multiple Access Channel With Heterogeneous Traffic and an Energy Harvesting Node," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 662–667.

[95] I. Krikidis, "Average Age of Information in Wireless Powered Sensor Networks," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 628–631, 2019.

[96] Y. Gu, H. Chen, Y. Zhou, Y. Li, and B. Vucetic, "Timely Status Update in Internet of Things Monitoring Systems: An Age-Energy Tradeoff," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5324–5335, 2019.

[97] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Minimizing the Age of Information in Broadcast Wireless Networks," in *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sept 2016, pp. 844–851.

[98] S. K. Kaul and R. Yates, "Status Updates Over Unreliable Multiaccess Channels," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jun. 2017, pp. 331–335.

[99] Y. Sang, B. Li, and B. Ji, "The Power of Waiting for More Than One Response in Minimizing the Age-of-Information," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–6.

[100] E. Najm and E. Telatar, "Status Updates in a Multi-stream M/G/1/1 Preemptive Queue," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2018, pp. 124–129.

[101] Z. Jiang, B. Krishnamachari, X. Zheng, S. Zhou, and Z. Miu, "Decentralized Status Update for Age-of-Information Optimization in Wireless Multiaccess Channels," in *isit*, June 2018, pp. 2276–2280.

[102] R. D. Yates and S. K. Kaul, "The Age of Information: Real-Time Status Updating by Multiple Sources," *IEEE Transactions on Information Theory*, vol. 65, no. 3, pp. 1807–1827, 2019.

[103] M. Moltafet, M. Leinonen, and M. Codreanu, "On the Age of Information in Multi-Source Queueing Models," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 5003–5017, 2020.

[104] Z. Jiang, B. Krishnamachari, S. Zhou, and Z. Niu, "Can Decentralized Status Update Achieve Universally Near-Optimal Age-of-Information in Wireless Multiaccess Channels?" in *International Teletraffic Congress ITC 30*, September 2018. [Online]. Available: http://arxiv.org/abs/1803.08189

[105] Y.-P. Hsu, "Age of Information: Whittle Index for Scheduling Stochastic Arrivals," in *isit*, June 2018, pp. 2634–2638.

[106] N. Lu, B. Ji, and B. Li, "Age-based Scheduling: Improving Data Freshness for Wireless Real-Time Traffic," in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. Mobihoc '18. New York, NY, USA: ACM, 2018, pp. 191–200. [Online]. Available: http://doi.acm.org/10.1145/3209582.3209602

[107] R. Talak, I. Kadota, S. Karaman, and E. Modiano, "Scheduling Policies for Age Minimization in Wireless Networks with Unknown Channel State," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2564–2568.

[108] R. Yates, "Status Updates Through Networks of Parallel Servers," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jun. 2018, pp. 2281–2285.

[109] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Minimizing the Age of Information Through Queues," *IEEE Transactions on Information Theory*, vol. 65, no. 8, pp. 5215–5232, 2019.

[110] A. M. Bedewy, Y. Sun, and N. B. Shroff, "The Age of Information in Multihop Networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1248–1257, 2019.

[111] Y. Sun, E. Uysal-Biyikoglu, and S. Kompella, "Age-optimal Updates of Multiple Information Flows," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2018, pp. 136–141.

[112] A. Maatouk, Y. Sun, A. Ephremides, and M. Assaad, "Status Updates With Priorities: Lexicographic Optimality," in *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*. IEEE, 2020, pp. 1–8.

[113] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing Data Freshness, Throughput, and Delay in Multi-Server Information-Update Systems," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, 2016, pp. 2569–2574.

[114] R. Talak, S. Karaman, and E. Modiano, "Minimizing Age-of-Information in Multi-Hop Wireless Networks," in *55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2017, pp. 486–493.

[115] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Age-Optimal Information Updates in Multihop Networks," *CoRR*, vol. abs/1701.05711, pp. 576–580, June 2017. [Online]. Available: http://arxiv.org/abs/1701.05711

[116] Q. He, D. Yuan, and A. Ephremides, "Optimal Link Scheduling for Age Minimization in Wireless Systems," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5381–5394, July 2018.

[117] R. Talak, S. Karaman, and E. Modiano, "Optimizing Information Freshness in Wireless Networks Under General Interference Constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 15–28, 2019.

[118] R. Talak, S. Karaman, and E. Modiano, "Optimizing Age of Information in Wireless Networks with Perfect Channel State Information," in *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE, 2018, pp. 1–8.

[119] C.-H. Tsai and C.-C. Wang, "Age-of-Information Revisited: Two-Way Delay and Distribution-Oblivious Online Algorithm," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 1782–1787.

[120] C.-H. Tsai and C.-C. Wang, "Jointly Minimizing AoI Penalty and Network Cost Among Coexisting Source-Destination Pairs," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 3255–3260.

[121] A. Arafa, K. Banawan, K. G. Seddik, and H. V. Poor, "Sample, Quantize, and Encode: Timely Estimation Over Noisy Channels," *IEEE Transactions on Communications*, vol. 69, no. 10, pp. 6485–6499, 2021.

[122] M. Klügel, M. H. Mamduhi, S. Hirche, and W. Kellerer, "AoI-Penalty Minimization for Networked Control Systems With Packet Loss," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 189–196.

[123] J. Pan, A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing Sampling for Data Freshness: Unreliable Transmissions With Random Two-Way Delay," *arXiv preprint arXiv:2201.02929*, 2022.

[124] E. T. Ceran, D. Gündüz, and A. György, "Average Age of Information With Hybrid ARQ Under a Resource Constraint," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.

[125] E. T. Ceran, D. Gündüz, and A. György, "Average Age of Information With Hybrid ARQ Under a Resource Constraint," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1900–1913, 2019.

[126] E. Najm, R. Yates, and E. Soljanin, "Status updates through M/G/1/1 queues with HARQ," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jun. 2017, pp. 131–135.

[127] C. Kam, S. Kompella, and A. Ephremides, "Experimental Evaluation of the Age of Information Via Emulation," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015, pp. 1070–1075.

[128] E. Sert, C. Sönmez, S. Baghaee, and E. Uysal-Biyikoglu, "Optimizing Age of Information on Real-Life TCP/IP Connections Through Reinforcement Learning," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, pp. 1–4.

[129] C. Sönmez, S. Baghaee, A. Ergişi, and E. Uysal-Biyikoglu, "Age-of-Information in Practice: Status Age Measured Over TCP/IP Connections Through WiFi, Ethernet and LTE," in *IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2018.

[130] H. B. Beytur, S. Baghaee, and E. Uysal, "Measuring Age of Information on Real-Life Connections," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2019, pp. 1–4.

[131] T. Shreedhar, S. Kaul, and R. Yates, "ACP: An End-to-End Transport Protocol for Delivering Fresh Updates in the Internet-of-Things," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1811.03353

[132] I. Kadota, M. S. Rahman, and E. Modiano, "Wifresh: Age-of-Information From Theory to Implementation," in *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021.

[133] U. Guloglu, S. Baghaee, and E. Uysal, "Evaluation of Age Control Protocol (ACP) and ACP+ on ESP32," in *2021 17th International Symposium on Wireless Communication Systems (ISWCS)*, 2021, pp. 1–6.

[134] H. B. Beytur, S. Baghaee, and E. Uysal, "Towards AoI-Aware Smart IoT Systems," in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 353–357.

[135] E. Uysal, O. Kaya, S. Baghaee, and H. B. Beytur, "Age of Information in Practice," *arXiv preprint arXiv:2106.02491*, 2021.

[136] AWS, "Amazon Elastic Compute Cloud ec2," https://aws.amazon.com/ec2/, 2022.

[137] Linux manual, "traceroute," https://linux.die.net/man/8/traceroute.

[138] "Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT)," https://www.orbit-lab.org/.

[139] Linux Manual, "hostapd - IEEE 802.11 AP," https://manpages.debian.org/testing/hostapd/hostapd.8.en.html.

[140] Linux manual, "iwconfig," https://linux.die.net/man/8/iwconfig.

[141] A. S. Tanenbaum, *Computer Networks*. Pearson Education India, 2002.

[142] R. K. Jain, D.-M. W. Chiu, W. R. Hawe *et al.*, "A Quantitative Measure of Fairness and Discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[143] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A First Look at QUIC in the Wild," in *International Conference on Passive and Active Network Measurement*. Springer, 2018, pp. 255–268.

[144] L. Kleinrock, "Internet Congestion Control Using the Power Metric: Keep the Pipe Just Full, But No Fuller," *Ad hoc networks*, vol. 80, pp. 142–157, 2018.

[145] Network Working Group, "RFC 5681: TCP Congestion Control," https://tools.ietf.org/html/rfc5681, 2009.

[146] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS operating systems review*, 2008.

[147] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proceedings of the conference on Communications architectures, protocols and applications*, 1994, pp. 24–35.

[148] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: Yet Another Highspeed TCP," in *Proceedings of PFLDnet*, vol. 7, 2007, pp. 37–42.

[149] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control (TCP)," RFC 5681, Sep. 2009. [Online]. Available: https://rfc-editor.org/rfc/rfc5681.txt

[150] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, Dec. 2003. [Online]. Available: https://rfc-editor.org/rfc/rfc3649.txt

[151] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, p. 83–91, Apr. 2003. [Online]. Available: https://doi.org/10.1145/956981.956989

[152] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport Over Wireless Links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001, pp. 287–297.

[153] S. Mascolo, L. Grieco, R. Ferorelli, P. Camarda, and G. Piscitelli, "Performance Evaluation of Westwood+ TCP Congestion Control," *Performance Evaluation*, vol. 55, no. 1, pp. 93–111, 2004, internet Performance Symposium (IPS 2002). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166531603000981

[154] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *IEEE INFOCOM 2004*, vol. 4. IEEE, 2004, pp. 2514–2524.

[155] C. Jin, D. Wei, S. H. Low, J. Bunn, H. D. Choe, J. C. Doylle, H. Newman, S. Ravot, S. Singh, F. Paganini *et al.*, "FAST TCP: From Theory to Experiments," *IEEE network*, vol. 19, no. 1, pp. 4–11, 2005.

[156] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. IEEE, 2017, pp. 215–218.

[157] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.

[158] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-Host Congestion Control for TCP," *IEEE Communications Surveys Tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[159] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *Proceedings-IEEE INFOCOM*, 2006.

[160] C. P. Fu and S. C. Liew, "TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks," *IEEE Journal on selected areas in communications*, vol. 21, no. 2, pp. 216–228, 2003.

[161] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks," *Performance Evaluation*, vol. 65, no. 6-7, pp. 417–440, 2008.

[162] Linux manual, "iPerf3," https://manpages.ubuntu.com/manpages/xenial/man1/iperf3.1.html.

[163] S. K. Kaul and R. D. Yates, "Age of Information: Updates With Priority," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2644–2648.

[164] A. Maatouk, M. Assaad, and A. Ephremides, "Age of Information With Prioritized Streams: When to Buffer Preempted Packets?" in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 325–329.

[165] J. Xu and N. Gautam, "Peak Age of Information in Priority Queuing Systems," *IEEE Transactions on Information Theory*, vol. 67, no. 1, pp. 373–390, 2021.

[166] M. Moltafet, M. Leinonen, and M. Codreanu, "Average AoI in Multi-Source Systems With Source-Aware Packet Management," *IEEE Transactions on Communications*, 2021.

[167] L. Huang and E. Modiano, "Optimizing Age-of-Information in a Multi-Class Queueing System," in *2015 IEEE International Symposium on Information Theory (ISIT)*, 2015, pp. 1681–1685.

[168] E. Najm and E. Telatar, "Status Updates in a Multi-Stream M/G/1/1 Preemptive Queue," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 124–129.

[169] E. Najm, R. Nasser, and E. Telatar, "Content Based Status Updates," *IEEE Transactions on Information Theory*, vol. 66, no. 6, pp. 3846–3863, 2020.

[170] F. Baker, J. Babiarz, and K. H. Chan, "Configuration Guidelines for DiffServ Service Classes," RFC 4594, Aug. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4594.txt

[171] "Differentiated Services Field Codepoints (DSCP)," https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml.

[172] "QoS: Classification Configuration Guide," https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_classn/configuration/xe-16/qos-classn-xe-16-book/qos-classn-ntwk-trfc.html, 2018.

[173] A. Malik, J. Qadir, B. Ahmad, K.-L. Alvin Yau, and U. Ullah, "QoS in IEEE 802.11-based Wireless Networks: A Contemporary Review," *Journal of Network and Computer Applications*, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515000892

[174] T. Szigeti, J. Henry, and F. Baker, "Mapping Diffserv to IEEE 802.11," RFC 8325, Feb. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8325.txt

[175] T. Høiland-Jørgensen, D. Täht, and J. Morton, "Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways," in *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2018.

[176] "tc-cake(8) — Linux Manual Page," https://man7.org/linux/man-pages/man8/tc-cake.8.html.

[177] Internet Engineering Task Force (IETF), "Multipath Extension for QUIC," https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/, 2022.

[178] L. Lo Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[179] Internet Engineering Task Force (IETF), "Deterministic Networking Architecture," https://www.hjp.at/doc/rfc/rfc8655.html, 2019.

[180] N. Mohan, T. Shreedhar, A. Zavodavoski, O. Waltari, J. Kangasharju, and S. K. Kaul, "(Poster) Redesigning MPTCP for Edge Clouds," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 675–677. [Online]. Available: 10.1145/3241539.3267738

[181] F. Michel, M. Trevisan, D. Giordano, and O. Bonaventure, "A first look at starlink performance," in *Proceedings of the 22nd ACM Internet Measurement Conference*, ser. IMC '22. New York, NY, USA: Association for Computing Machinery, 2022.

[182] X. Yuan, M. Wu, Z. Wang, Y. Zhu, M. Ma, J. Guo, Z.-L. Zhang, and W. Zhu, "Understanding 5g performance for real-world services: A content provider's perspective," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 101–113. [Online]. Available: https://doi.org/10.1145/3544216.3544219

[183] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao, F. Qian, and Z.-L. Zhang, "A variegated look at 5g in the wild: Performance, power, and qoe implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 610–625. [Online]. Available: https://doi.org/10.1145/3452296.3472923