ACP+: An Age Control Protocol for the Internet

Tanya Shreedhar†, Sanjit K. Kaul†, and Roy D. Yates‡, Fellow, IEEE

Abstract—We present the age control protocol ACP+, a transport layer protocol that regulates the rate at which update packets carrying information from a source are sent over the Internet to a monitor. The source would like to minimize the average age of information at the monitor. Extensive experimentation helps shed light on age control over the current Internet and its implications for sources sending updates over a shared wireless access to monitors in the cloud. Surprisingly, age minimizing rates over fast Internet paths are about 0.5 Mbps, which is a small fraction, for example, of link rates supported by WiFi wireless access technology. We also show that congestion control algorithms employed by the Transmission Control Protocol (TCP), including hybrid approaches that achieve higher throughputs at lower delays than traditional loss-based congestion control, are unsuitable for age control.

Index Terms—Age control protocol, age, freshness, Internet Protocols, congestion control, Internet of Things.

I. INTRODUCTION

The availability of inexpensive embedded devices with the ability to sense, communicate, and actuate, has led to a proliferation of real-time cyber physical system applications spanning domains such as health care, smart homes, transportation and natural environment monitoring. IoT devices are deployed alongside users in homes/offices/cities and connect to the Internet over wireless last-mile access. 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 includes a more current measurement. These updates are communicated over the network to a remote server in the cloud, which processes them for analytics and/or to compute any required actuation. As a result, the devices may receive a sequence of updates containing actuation commands.

For such applications, it is desirable that freshly sensed measurements are available at servers and freshly generated actuation commands are available at the devices. We will refer to the recipient and sender of an update as monitor and source, respectively. An update packet contains information (for example, measurements, actuation commands) sent by a source to a monitor. It also contains a timestamp, which is the time of generation of the information sent by the source.

At time $t$, the freshness of a source’s information is measured by its instantaneous age $\Delta(t)$ at a monitor. $\Delta(t)$ is the time elapsed since generation of the update with the most recent timestamp received by the monitor until $t$. When a stream of updates is delivered, it is desirable to minimize the average age of information, referred to as AoI or simply age, at the monitor. Specific examples include the problem of sampling a Wiener process (source) to minimize the expected mean-squared error at a monitor, which reduces to that of minimizing the average age [1] when the sampling policy is unaware of the process. When estimating the state of a linear time-invariant system at a monitor, under certain conditions, minimizing the estimation error is same as minimizing average age, when sampling is independent of the state [2].

While one may achieve a low packet delay by simply choosing a low rate for the source to send updates, this may be detrimental to freshness; a low update rate can lead to a large age at the monitor because of infrequent updates from the source. On the other hand, sending updates at a high rate over the network can also be detrimental to freshness if each received update has high age due to large network delays. In the absence of network mechanisms that mitigate congestion, freshness at a monitor is optimized by the source smartly choosing an update rate [3], [4], [5], [6], [7] matched to the network.

Figure 1 depicts the typical behavior of the metrics of delay and age as a function of throughput, which has been observed for a wide variety of systems that are subject to congestion effects [7]. Observe that there exists a sending rate (and corresponding throughput) at which age is minimized.
resilient than throughput-intensive applications such as file transfer. In the former, retransmitting an old update is typically not beneficial [9]. However, the latter require reliable transport and are relatively delay tolerant. They often use a variant of Transmission Control Protocol (TCP) for end-to-end delivery of application packets.

As we show in Section V and later in Section XI, 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.

In this paper, we detail the age control protocol ACP+, which builds on an early version named ACP [10], [11] that was the first proposal for a transport layer age control protocol. ACP+ was introduced in [12], which also details the differences between ACP+ and ACP, the failings of ACP in shared access settings, and provides insights on the age control performance of TCP congestion control algorithms. ACP+ tries to keep the average age of information at the monitor to a minimum. To this end, ACP+ adapts the rate of updates from a source, in a network transparent manner, to perceived congestion over the end-to-end path between the source and the monitor. Consequently, like ACP, it also limits congestion that would otherwise be introduced by sources sending to their monitors at unnecessarily fast update rates.

Our contributions in this paper include the following (a) We define the age control problem over the Internet and intuit a good age control behavior using a mix of analysis and simulations. (b) We evaluate the suitability of TCP and UDP for sending ACP+ packets over the Internet. Over a range of fixed source update rates, we investigate the impact of different TCP configurations, such as congestion window and segment sizes, on age at a monitor, and compare with UDP. (c) We detail how ACP+ interfaces with the TCP/IP networking stack via UDP and with sources and monitors. We describe the ACP+ control algorithm. (d) 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. These experiments have many sources sharing a WiFi access (with contention ranging from low to moderately high) followed by many hops to a monitor over a fast Internet backbone. (e) We shed light on age control over end-to-end paths in the current Internet. The age optimizing rate when a source updates a monitor over a path that includes a WiFi access followed by the Internet backbone 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 backbone rates as high as 200 Mbps. In fact, it is also the age optimizing rate over the path in the absence of a first WiFi hop. The intercontinental path, much faster than the WiFi link, turns out to be the constraining factor with respect to the achievable age over the end-to-end path, likely because of the other traffic flows that utilize it. We observe that at the age optimal rate, depending on the network, a source may send multiple updates per round-trip-time (RTT) or may send an update over many RTT. 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 is in sharp contrast to congestion control algorithms that attempt to maximize throughput. For all such algorithms it is sufficient to saturate the bottleneck link, with differences in algorithms being about their achieved RTT performance and what is used as an indicator of the bottleneck link getting close to saturation (increased RTT, packet drops, or a mix).

We investigate age, throughput and delay trade-offs obtained when using state-of-the-art TCP congestion control algorithms to regulate the rate of updates over the Internet. We experiment with a mix of loss-based (Reno [13] and CUBIC [14]), delay-based (Vegas [15]) and hybrid congestion control algorithms (YeAH [16] and BBR [17]) for different settings of receiver buffer size. We conclude that TCP congestion control algorithms are unsuitable for age control. In fact, as contention on the access network increases, the AoI performance of TCP degrades unacceptably.

The rest of the paper is organized as follows. In the next section, we describe related works. In §III, we define the age control problem. In §IV, we use simple queueing models to intuit a good age control protocol and discuss a few challenges. In §V, we compare TCP and UDP and show why UDP is better suited to transport update packets of ACP over the Internet. We detail the Age Control Protocol, how it interfaces with a source and a monitor, and the protocol’s sequence diagram in §VI. §VII details the control algorithm that is a part of ACP+. This is followed by real-world evaluation over Intercontinental paths and a contended WiFi access in §VIII. We discuss simulation setup and results in §IX. We discuss the various congestion control schemes used in the Internet in §X and ageing over the Internet using these schemes in §XI. In §XII, we add a discussion on issues of efficiency, fairness, starvation and friendliness, before concluding the paper.

II. RELATED WORK

Various works, [2], [3], [4], [6], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27] have analyzed queue theoretic abstractions of networks, under different assumptions about update arrival processes, service distributions, number of servers and sources, and queue management like prioritization and preemption. The analysis results in distributional properties of age at the monitor or, more typically, the expected value of the time-average age or the peak age. Such works can help choose an appropriate arrival rate for the one or more sources that are sending packets through the chosen queuing system. The choice of rate is one-shot and doesn’t adapt to current network conditions.

Works have considered scheduling of updates for multiple sources sharing a communications network [28], [29], [30], [31]. The scheduling of updates for parallel server systems was considered in [32], [33], [34], [35], and [36]. AoI analysis for

---

1 The sources and monitors stay unaware of the network.

2 We will use ACP to refer to the protocol when the features being described in ACP+ are the same as in ACP. We will use ACP+ exclusively when describing the control algorithm, the simulations, and the experiments.
multiple hop and multiple source networks has also received attention in [33] and [37].

There are works that investigate sampling policies to optimize AoI [38], [39], [40], [41], [42]. 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 [38], [39]. The optimal sampling policy in such cases is a threshold one, either deterministic or randomized. Sampling policies for unreliable transmissions are considered in [40] and [41]. Recently, [42] proposed 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 transmission. Otherwise, the source sends a new update immediately without waiting. These works minimize age in the context of a stop and wait protocol. 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 [43] explored practical issues such as contention window sizes, the subsequent AoI literature has primarily been focused on analytically tractable 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 path between source and monitor. It is similar in spirit to hybrid ARQ (Automatic Repeat Request) based updating schemes [44], [45] that learn the wireless channel. 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.** There is limited systems research on ageing of information and its optimization in real-world networks [10], [11], [12], [46], [47], [48], [49], [50]. Various works on real networks can be found in [51]. In [46], authors discuss 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 key takeaway from that work is the need for an AoI optimizer that can adapt to changing network topologies and delays. Our previous work proposes the Age Control Protocol (ACP) [10], [11], [47]. ACP attempts to minimize the AoI of a source at a monitor connected via an end-to-end path over the Internet. In [12], we first proposed ACP+ as a modification to ACP and compared it with other state-of-the-art TCP congestion control algorithms used in the Internet. In [48], WiFresh, a MAC and application-layer solution to ageing of updates over a wireless network is proposed. While both [11] and [48] 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 [50], we study the coexistence of age-sensitive ACP+ flows and throughput-hungry TCP flows sharing an end-to-end Internet path over a WiFi access. We found that 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.

In summary, unlike other works, ACP and ACP+ aim to provide a practical age control algorithm that seeks 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 on the other sources that have access to it.

### III. The Age Control Problem

In this section, we define the age control problem. To simplify the presentation, 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 \( t \).

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 2.

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. 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 \( d_i \).

We want to choose the rate \( \lambda \) (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 \( \lambda \) to nonstationarities in the network.

### IV. Good Age Control Behavior and Challenges

ACP must suggest a rate \( \lambda \) 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 first-come-first-served (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 will 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 $\lambda$ 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. 3 In fact, such a rate $\lambda$ is age minimizing even when updates pass through $Q$ such queues in tandem (series), where $Q > 1$. The update is received by the monitor when it leaves the last queue in the sequence. The rate $\lambda$ 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, we will refer to the number of updates from a source that are in the network as its backlog. For the $Q$ queues in tandem the backlog will be $Q$, with one update in each server. See Figure 3.

Of course, the assumed network is a gross idealization. We assumed a series of identical constant service time facilities and that the time spent in service and instant of service completion were known exactly. We also assumed 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 control algorithm, we make a few salient observations about the impact of non-stationary network conditions on average age and the resulting challenge of keeping it small. We build intuition about how network queues work conditions on average age and the resulting challenge of keeping it small. We build intuition about how network queues work conditions on average age and the resulting challenge of keeping it small.

Figure 3. An illustration of queue occupancy and its impact on age for a network of identical queues with deterministic service in tandem. Update at the head of a queue is in service.

While such just-in-time updating is optimal when the service is deterministic, the same is not true for general service distributions [38], [39].
smaller than the system time. The implication being that on
an average it is optimal to send slightly more than one (≈ 1.12,
obtained by dividing the system time by the inter-arrival time
in the pair of values marked by + on the µ = 1 curve) 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).

Figure 4c shows how the optimal average backlog varies
for the two queues in tandem, as a function of the service
rate µ2 with µ1 = 1. For each resulting pair of service rates,
the age-optimal arrival rate λ∗ is selected. For the fixed µ1,
λ∗ increases as µ2 increases, and this causes the average
backlog in queue 1 to increase. However, as λ∗ increases
more slowly than µ2, 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 queue 2, the optimal λ∗ must adapt to
the bottleneck queue.

These observations stay the same on swapping µ1 and µ2.
Moreover, when the rates µ1 and µ2 are similar, the queues
see similar backlogs. Notably, when µ1 = µ2 = 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.

On the impact of a larger number of hops: To see if
this intuition generalizes beyond two 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 illustrated in Figure 5.
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
another traffic generator in the network that randomly gen-
erates packets at the source that are sent to the monitor, and
occupies, on an average, 0.2 Mbps of the path from the source
to the monitor. The different configurations are summarized in
Table 1. For each network configuration we have the source
send updates over UDP to the monitor using an a priori chosen
rate λ (fixed inter-update arrival time of 1/λ). We vary λ over
a range of values and for each λ we calculate the obtained
time-average age. We empirically identify the age minimizing
λ for each given network.

Figure 6 shows the time-average backlog (queue occupancy)
at the different nodes in the network at the age-optimal
arrival rate λ∗. The backlog at a node includes any packet
being transmitted on a node’s outgoing P2P link or 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 E has links much faster than that
of Net A, for both these networks the average backlog at
all nodes is close to 1. It is smaller than 1 because of the other
0.2 Mbps traffic flow that shares the network. Recall that this
flow also originates at the source. This explains why the source
sees a slightly larger average queue backlog.

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 Figure 6, similar observations apply to Net C and Net D. In summary, at the age-minimizing update rate $\lambda$, links that are relative bottlenecks each see an average backlog of no more than one update packet. Nodes with faster links see smaller backlogs in proportion to how fast their links are relative 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).

V. UDP VS. TCP FOR TRANSPORTING ACP UPDATES

While ACP adapts the rate of updates from a source to a monitor, the updates must be sent over the Internet (an IP network). We compare the effectiveness of TCP and UDP, two commonly used transport protocols in the TCP/IP stack, as a substrate for transporting ACP updates. We compare the two over a range of fixed source rates of updates and show why UDP is a better choice than TCP, for any fixed rate.

We simulated a simple network consisting of a source that sends 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 send its update packets over the TCP connection, for an a priori chosen fixed number of updates generated every second. We compare the obtained age with when UDP is used instead of TCP. We perform simulations for different choices of rate of update generation by the source. A larger utilization of the 1 Mbps links is achieved by the source generating update packets at a faster rate.

We calculate utilization as the product of the number of bits in an update and the rate at which updates are generated by the source, normalized by 10$^6$ bits/sec.

Retransmissions and In-order Delivery: Figure 7 illustrates the impact of packet errors on TCP. A packet was dropped independently of other packets with probability 0.1. The update packet size was set to 536 bytes. The figure compares the average age at the monitor and the average update packet delay, which is the time elapsed between generation of a packet at the source and its delivery at the monitor, when using TCP and UDP. Under TCP, the time-average age achieves a minimum value of 0.185 seconds when the source utilizes a fraction 0.1 of the available 1 Mbps to send update packets. This is much larger than the minimum age of $\approx 0.01$ seconds at a UDP utilization of $\approx 0.9$. The large minimum age when using TCP is explained by the way TCP guarantees 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 8 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. ACP will use UDP to provide update packets with end-to-end transport over IP (§VI).

TCP Congestion Control and Small Packets: Next, we describe the impact of small packets, smaller than the minimum sender maximum segment size (SMSS) bytes, on the TCP congestion algorithm and age. This is especially relevant to a source sending 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. The minimum SMSS is 536 bytes. Observe in the upper plot of Figure 9 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
SMSS bytes are acknowledged. TCP does this to optimize the overheads associated with sending payload. Packets with fewer bytes may 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 9. Smaller packets wait longer in the TCP send buffer before they are sent out by the TCP sender, which explains the larger age in Figure 9.

VI. 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 10 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 of updates generated by the sources.

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 should generate updates. To be able to calculate the suggested 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 ACP packet in return. The ACP 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 information update from the source.

Figure 11 shows the sequence diagram 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 is 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.

VII. THE ACP+ CONTROL ALGORITHM

Let the control epochs of ACP+ be indexed 1, 2, . . ., such that epoch k starts at time \( t_k \) and ends at \( t_{k+1} \), which is the beginning of the next epoch. Let \( \lambda_k \) be the update rate set by the control algorithm for epoch k. The source ACP+ transmits updates at a fixed time period of \( 1/\lambda_k \) seconds in the interval \( (t_k, t_{k+1}) \). For the first control epoch, beginning at \( t_1 \), the update rate \( \lambda_1 \) is set to the inverse of the average packet round-trip-times (RTT) obtained at the end of the initialization phase.

The length \( T_k = t_{k+1} - t_k \) of the \( k \)th control epoch is set as an integral number \( \eta \) of time periods, each of length \( 1/\lambda_k \). That is \( T_k = \eta/\lambda_k \). A control epoch should be long enough to allow for reasonable estimates of the time-average age and backlog that result from setting the update rate to \( \lambda_k \) for epoch \( k \). In this work, we find \( \eta = 10 \) to work well for all our chosen scenarios in simulations and real-world experiments. An ACP+ source sends updates indexed \( i = 1, 2, . . . \) with the \( i \)th update having timestamp \( a_i \). From Section III, \( d_i \) is the time the update is received by the monitor. Let \( \hat{d}_i \) be the time the ACP corresponding to it is received by the source.

At the source, the ACKs received in sequence are used to construct approximations \( \hat{\Delta}(t) \) and \( \hat{B}(t) \), respectively, of the age process \( \Delta(t) \) at the monitor and the backlog process \( B(t) \) that tracks the number of update packets currently in the network. To estimate \( B(t) \), we define

\[
S(t) = \max\{i: a_i \leq t\},
\]

\[
\hat{N}(t) = \max\{i: \hat{d}_i \leq t\}.
\]

At time \( t \), \( S(t) \) is the index of the freshest update that has been transmitted and \( \hat{N}(t) \) is the index of the freshest update for which an in-sequence ACK has been received at the source. The estimated backlog at time \( t \) is

\[
\hat{B}(t) = S(t) - \hat{N}(t).
\]

In (2), the estimated backlog increases by 1 when the source sends a new update at time \( t = a_i \). Further, out-of-sequence ACKs, which the source discards, won’t change \( \hat{N}(t) \) and won’t reduce the estimated backlog. Also, when the ACK corresponding to an update \( i \) is received in sequence, update \( i \) as well as any unacknowledged updates older than \( i \) are removed from the estimated backlog. This estimated backlog
undercounts updates that arrive out-of-sequence at the monitor. In the absence of out-of-sequence updates, it is a conservative estimate in that updates are removed from the backlog only when ACKs are returned to the source.

To estimate the age process at the source, we define

$$\hat{\Delta}(t) = t - \alpha \hat{N}(t).$$

(3)

The effect of (3) is that only in-sequence ACKs reduce the estimated age $\hat{\Delta}(t)$. Moreover, if the ACK for update $i$ is received in sequence, then $\hat{\Delta}(d_i) = \hat{d}_i - a_i$, which is the RTT corresponding to update $i$. Resetting the estimate of age to the RTT overestimates the true age $d_i - a_i$ of the update packet, when it was received at the monitor, by the time taken to send the ACK from the monitor to the source. However, this estimate is useful in practice as it doesn’t require any time synchronization between the ACP+ source and monitor. Further, the overestimation offsets the time-average age by the one-way delay from the monitor to the source, suffered by the ACK, and doesn’t affect the update rate that minimizes the average age. This has also been shown in [51].

The ACP+ source also maintains exponentially weighted moving averages (EWMAs) RTT and $\hat{Z}$ of the RTT and the ACK interarrival time. When an ACK is received in sequence, ACP+ calculates the corresponding average of the number of backlogged packets. The EWMAs RTT and $\hat{Z}$ are updated at $t = d_i$ as

$$\text{RTT}(t) = (1 - \alpha)\text{RTT}(t^-) + \alpha \text{RTT}_i,$$

(6a)

$$\hat{Z}(t) = (1 - \alpha)\hat{Z}(t^-) + \alpha \hat{Z}_i.$$  

(6b)

In the absence of an in-sequence ACK, RTT$(t)$ and $\hat{Z}(t)$ remain unchanged.

Let $\Delta_k$ be the estimate at the ACP+ source at time $t_k$ of the time-average update age at the monitor. It is obtained by calculating the time-average of the estimated age function $\hat{\Delta}(t)$ over $(t_{k-1}, t_k)$. Similarly, define $\bar{B}_k$ to be the time-average of backlog $\hat{B}(t)$ calculated over the interval $(t_{k-1}, t_k)$. At time $t_k$, the source ACP+ calculates the difference $\delta_k = \Delta_k - \Delta_{k-1}$ in average age measured over epochs $(t_{k-1}, t_k)$ and $(t_{k-1}, t_{k-2})$. Similarly, it calculates $b_k = \bar{B}_k - \bar{B}_{k-1}$. ACP+ at the source chooses an action $u_k$ at the beginning of the $k^{th}$ epoch that targets a change $b^*_{k+1}$ in average backlog over the interval of length $T_k$ of the $k^{th}$ epoch. The actions may be broadly classified into additive increase (INC), additive decrease (DEC), and multiplicative decrease (MDEC). MDEC corresponds to a set of actions $\{\text{MDEC}(\gamma) : \gamma = 1, 2, \ldots\}$. Specifically,

$$\text{INC: } b^*_{k+1} = 1,$$

(7a)

$$\text{DEC: } b^*_{k+1} = -1,$$

(7b)

$$\text{MDEC}(\gamma): b^*_{k+1} = -(1 - 2^{-\gamma})B_k.$$  

(7c)

ACP+ attempts to achieve $b^*_{k+1}$ by setting $\lambda_k$ appropriately. The estimate $\hat{Z}$ at the source ACP+ of the average inter-update arrival time at the monitor gives us the rate $1/\hat{Z}$ at which updates sent by the source currently arrive at the monitor. The change in the average backlog that will result from setting the rate to $\lambda_k$ is $(\lambda_k - 1/\hat{Z})\hat{Z}$. Therefore, a desired change of $b^*_{k+1}$ requires choosing

$$\lambda_k = \frac{1}{\hat{Z}} + \frac{b^*_{k+1}}{\text{RTT}}.$$  

(8)

Note that $\hat{Z}$ and RTT are time-averages. $(1/\hat{Z})\text{RTT}$ is the corresponding average of the number of backlogged packets. Assuming that an epoch is long enough, $\lambda_k\hat{Z}$ gives us the average number of backlogged packets in case a rate of $\lambda_k$ is used. In (8) we assume that any change in RTT on setting the rate to $\lambda_k$ can be ignored.

---

**Algorithm 1** ACP+ Control Algorithm

1: **INPUT:** $b_k$, $\delta_k$
2: **INIT:** $\text{flag} \leftarrow 0$, $\gamma \leftarrow 0$
3: while true do
4: if $b_k > 0$ 
5: if $\text{flag} = 1$ then
6: $\gamma = \gamma + 1$
7: $\text{MDEC}(\gamma)$: $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
8: else
9: $\text{DEC: } b^*_{k+1} = -1$
10: $\text{flag} \leftarrow 1$
11: if $b_k < 0$ 
12: $\text{INC: } b^*_{k+1} = 1$
13: $\text{flag} \leftarrow 0$, $\gamma \leftarrow 0$
14: if $b_k < 0$ then
15: $\text{INC: } b^*_{k+1} = 1$
16: $\text{flag} \leftarrow 0$, $\gamma \leftarrow 0$
17: if $b_k < 0$ then
18: if $\text{flag} = 1$ 
19: $\text{MDEC}(\gamma)$: $b^*_{k+1} = -(1 - 2^{-\gamma})B_k$
20: else
21: $\text{DEC: } b^*_{k+1} = -1$
22: $\text{flag} \leftarrow 0$, $\gamma \leftarrow 0$
23: $\text{UPDATE~LAMBDA}(b^*_{k+1})$
24: wait $T_k$
25: **function** $\text{UPDATE~LAMBDA}(b^*_{k+1})$
26: $\lambda_k = \frac{1}{\hat{Z}} + \frac{b^*_{k+1}}{\text{RTT}}$
27: if $\lambda_k < 0.75 \times \lambda_{k-1}$ then
28: $\lambda_k = 0.75 \times \lambda_{k-1}$ > Minimum $\lambda$ threshold
29: else if $\lambda_k > 1.25 \times \lambda_{k-1}$ then
30: $\lambda_k = 1.25 \times \lambda_{k-1}$ > Maximum $\lambda$ threshold
31: return $\lambda_k$

---

6We calculate the area under the age function and divide it by the interval of time it spans. The area can be approximated as a sum of areas $(d_i - d_{i-1})\text{RTT}_{i-1} + 0.5 \times (d_i - d_{i-1})^2$ of quadrilaterals (see Figure 2), one for each update $i$ whose ACK is received by the source at $d_i$ during the epoch.
Algorithm 1 summarizes how ACP+ chooses its action $u_k$ as a function of $b_k$ and $\delta_k$. The source ACP+ targets a reduction in average backlog over epoch $k$ in case either $b_k, \delta_k > 0$, or $b_k, \delta_k < 0$. Both $b_k$ and $\delta_k$ being positive (line 4 in Algorithm 1) indicates that the update rate is such that updates are experiencing larger than optimal delays due to backlog in excess of the optimal, and thus larger than optimal age. To illustrate using the queuing systems in Figure 4a, both $b_k$ and $\delta_k$ being positive implies that the average age $\Delta$ is in the region seen to the right of the minimum, where $\lambda > \lambda^*$. In this region an increase in backlog coincides with an increase in age. We must reduce backlog to reduce age. ACP+ attempts to reduce 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 $\delta$ by 1 in line 6) attempt to decrease backlog even more aggressively, dividing it by a larger power of 2.\(^\text{7}\)

The condition that $b_k$ and $\delta_k$ are both negative (line 17) corresponds to a reduction in both age and backlog. With regards to Figure 4a, this implies that the average age $\Delta$ is in the region seen to the right of the minimum, where $\lambda > \lambda^*$. In this region, reduction in backlog coincides with reduction in age. ACP+ greedily aims at reducing backlog further in the hope that age will reduce too. It attempts MDEC (line 19) if previously the condition $b_k, \delta_k > 0$ was satisfied and MDEC was chosen; otherwise, it attempts the additive decrease DEC.

The source ACP+ targets an increase in average backlog over the next control epoch in case either $b_k > 0$, $\delta_k < 0$ or $b_k < 0, \delta_k > 0$. On the occurrence of the first condition (line 14) ACP+ greedily attempts to increase backlog. In Figure 4a, the first condition implies that the average age $\Delta$ is in the region seen to the left of the minimum, where $\lambda < \lambda^*$. It is here that an increase in backlog reduces age. The condition $b_k < 0, \delta_k > 0$ hints at too low an update rate causing an increase in age. In Figure 4a, this implies that the average age $\Delta$ is in the region seen to the left of the minimum, where $\lambda < \lambda^*$. It is here that a reduction in backlog increases age. So, ACP+ attempts an additive increase (line 13) of backlog.

VIII. Updates Over Intercontinental Paths and a Contended WiFi Access

Figure 12 illustrates our real-world experimental setup. We used the ORBIT testbed,\(^\text{8}\) 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. The dense spatial configuration of radio nodes allows emulation of high contention over WiFi. We configured multiple ORBIT nodes as updating sources and an additional ORBIT node as an 802.11n access point. This access point acts as a gateway to the Internet for our sources and is configured to operate at 5 GHz on a fixed channel and a fixed WiFi physical layer rate using hostapd and the iwconfig utility. Fixed WiFi rates, in contrast to allowing WiFi rate control, enable better understanding of the impact of the WiFi access and the Internet on age control. Our sources send updates to an EC2 AWS\(^\text{9}\) instance in Mumbai, India, which serves as our ACP+ monitor.

For our experiments, we selected up to 80 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 MAC 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, we configured only the wireless network within the ORBIT testbed; the rest of the path to AWS Mumbai traversed 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 alone sends to the monitor in a stop-and-wait fashion, i.e., 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. About 40% of it is speed-of-light propagation delays and the rest is time spent waiting in router queues and transmission over hops in the round trip path.

We perform at least five repeats\(^\text{10}\) of each 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 on different days of the week and at different times of the day. In all experimental results that follow in the paper, the average age is the time average of the estimated age (3). We also compared ACP+ with fixed rate UDP. A summary is in the Appendix B.

\(^7\)We discuss further the case when $B_k < 1$ and both $b_k$ and $\delta_k$ are positive. First, as described, ACP+ will choose DEC, which will attempt to reduce $B_k$ by 1. This of course is not possible as the backlog can’t be forced to less than 0. The setting may at most result in (8) giving a $\lambda_k < 0$. However, line 27 forces $\lambda_k$ to be set to 0.75 of its current value. In case MDEC is chosen post DEC, given $B_k < 1$, it will target a reduction in backlog less than 1, which is smaller than the reduction of 1 always chosen by DEC. Continued applications of MDEC will push the desired backlog closer to 0.

\(^8\)https://www.orbit-lab.org/

\(^9\)https://aws.amazon.com/

\(^10\)For smaller number of sources, a repeat lasts till the ACP+ sources send about 5000 updates. We increased this number to 50000 per source for when we have 80 sources. This ensures a long period of time during which all sources contend for the shared access.
of throughputs of the sources sharing it) normalized by the WiFi link rate. When the number of sources sharing the access is small and 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. Not surprisingly, the region of low contention extends to a larger number of sources for larger WiFi link rates.

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 use ORBIT to test ACP+ in scenarios wherein one would expect age control to backlog on an average less than an update per source per round-trip-time. We will resort to simulations to evaluate ACP+ in such scenarios. We simulate high WiFi contention together with small RTT values.

IX. Simulations Setup and Results

Figure 5 shows the network used for simulations. We performed experiments for 1 – 48 sources accessing AP-1 using the WiFi (802.11g) medium access. We simulated for sources spread uniformly and randomly over an area of 20 × 20 m². The channel between a source and AP-1 was chosen to be log-normally distributed with the standard deviation of 4, 8 or 12. The pathloss exponent was 3. We used the network simulator ns3 together with the YansWiFiPhyHelper. Our simulated network is however limited in the number of hops, which is six. WiFi physical (PHY) layer rate was set to 12 Mbps and that of the P2P links was set to 6 Mbps. The update size is 536 bytes with baseline RTT of 5.5 ms.

To compare the age control performance of ACP+, we use Lazy as defined in our earlier work [11]. 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 RTT every

A. Takeaways for Age Control in the Internet

For an update payload size of 1024 bytes, we observe in Figures 13a and 13c that ACP+ achieves a small average age (about the same as the baseline RTT), for a single source at an end-to-end throughput 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 small age-optimizing throughput has been observed by us over other Internet paths (see Section XI and earlier work [11]).

An age optimizing throughput much smaller than the access link rates enables multiple ACP+ sources to share the access without suffering an age penalty because of the other sources. Specifically, Figure 13a shows that the average age per source stays similar when there are 1 – 5 sources sharing a 6 Mbps WiFi access. The age also stays similar for 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. We see large RTT(s) (Figure 13d) and ACP+ maintains smaller backlogs (Figure 13b) of updates per source. We have a sharp reduction in per source throughputs, shown in Figure 13c.

Age-optimizing throughputs much smaller than access link rates have significant 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 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 gets large, saturation of the access link becomes the constraining factor with regards to age optimization.

Figure 14 illustrates the two regimes of age control. It plots the utilization of the access channel (measured as the sum
time an ACK is received and sets the current update rate to the inverse of RTT, targeting an average backlog of 1.

Figure 15a 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 15b. When we have just one source, ACP+ tries to fill each queue in the network with an update. Note that, unlike the Internet, this network isn’t shared by any other traffic. 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.

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 and contention free manner. Note that in the absence of contention and packet drops due to channel errors and given deterministic transmission times that are the same for updates from all sources, round-robin optimizes the average age [52, Lemma 4]. To see this, consider the simplified setting in which the WiFi and P2P links are the same and 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 WiFi access, which results in larger delays in WiFi hop.

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 [53] 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. For Lazy, the index is ≥ 0.98 for all settings. 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.

X. CONGESTION CONTROL APPROACHES IN THE INTERNET: DO WE NEED ACP?

TCP is the dominant protocol used in the Internet with ≈ 90% traffic share [54]. TCP congestion control is the primary mechanism by which end hosts share Internet bandwidth. For the purpose of TCP’s operation, the end-to-end path may be abstracted away as a link with bottleneck bandwidth $BW_{\text{Btl}}$ and a round-trip propagation time of $RTT_{\text{base}}$ (baseline RTT) [17]. Figure 16a provides an illustration, akin to that in [17, 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 $BW_{\text{Btl}}$, the TCP packets see a low and fixed RTT of $RTT_{\text{base}}$. Once the offered load becomes larger than $BW_{\text{Btl}}$, 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 loss-based congestion control allows for an increasing number of unacknowledged bytes from an application to flow through the network pipe until one or more bytes sent are lost. Loss of bytes signals congestion to the TCP sender and occurs because a link along the end-to-end path is operating in the buffer limited region (see Figure 16a). An end-to-end flow may 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 16a suggests that one would ideally like to operate at the lower “knee” in the curve, i.e., close to the bottleneck throughput $BW_{\text{Btl}}$ but 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) [17] 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. Intriguingly, this point of operation would satisfy the goal of age control by resulting in the highest rate of packet delivery at the monitor at the smallest possible packet delays. In fact this combination of a throughput of \( BW_{\text{full}} \) and round-trip times of \( RTT_{\text{base}} \) is achieved by the snapshot in Figure 3c that illustrates a good age control algorithm in action.

Of course, as was observed in [55] 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 \( BW_{\text{full}} \) could be much larger than \( RTT_{\text{base}} \). Figure 16b provides an illustration of \textit{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.

The shift in congestion control algorithms from keeping the pipe full to “keeping the pipe just full, but no fuller” [55], motivates this empirical study of how information at a monitor would age if updates were transmitted using the congestion control algorithms. We evaluate a mix of loss-based (Reno [13] and CUBIC [14]), delay-based (Vegas [15]) and hybrid congestion control algorithms (YeAH [16] and BBR [17]).

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 17, an application (source) 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 (monitor). 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 monitor (TCP 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 (a generate-at-will model [77]) to be incorporated in a segment just as TCP schedules it for sending.

We approximate the age of the segment when it arrives at the monitor 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 correspondingACKs. The RTT(s) and the inter-delivery times together allow us to come up with an estimate of the time-average of the age function (see Figure 2) at the receiver that results from a congestion control algorithm.

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. We focus on paths in the cloud, specifically between AWS data centers, and observe a very small percentage of loss. Also, this loss is not because of link transmission errors but because of buffer overflows in routers. The latter result in the process of congestion control estimating the bottleneck bandwidth.

XI. 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 the core network. 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 the Amazon Web Services (AWS). Next, we look at an end-to-end path in which the first hop is a shared and contended wireless access.

A. Ageing Over the Core Network

Figure 18 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\texttt{traceroute} that the underlying network between our two chosen instances was served by the AWS private WAN. For both the ACP+ and TCP experiments, we deployed the source (TCP sender) in AWS Frankfurt and the monitor (TCP receiver) in AWS Mumbai.

We used iPerf3, which is a well-known network bandwidth measurement tool for generating TCP traffic. The rate at which it sends TCP traffic is governed by the round-trip time and congestion control mechanisms of TCP. We use 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. We further ran different TCP receive buffer settings, as detailed later. Each run of the experiment lasted 200 s. Considering that end-to-end RTT is ≈ 110 ms in our setup, TCP spends a majority of the transfer time in the steady-state phase. Lastly, we observe that in our experiments the age performance of TCP congestion control algorithms does not take a hit because of TCP’s feature of guaranteed in-order delivery (recall simulations in Section V).\(^\text{15}\)

1) 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. TCP performance is often optimized by setting default and maximum values of the receiver buffer. This enables support of links with high bandwidth.\(^\text{16}\) 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 path doesn’t become the bottleneck.

2) Significant Queue Waiting Delays: Figure 19 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 contribute significantly to the time spent in router queues awaiting transmission. 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. Also note that while the segment lengths chosen by BBR-d1m3 and BBR-d5m5 are quite different, the throughputs (not shown) achieved by the two BBR configurations, averaged over all runs, are about the same (228 Mbps and 217 Mbps, respectively).

3) Delay Vs. Age: Figure 20 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 ≈ 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.

4) ACP+ Vs. BBR-d1m1: Before we delve further into the relative performances of ACP+ and BBR-d1m1, let’s consider Figure 21 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 throughput similar to BBR-d1m3. 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 (≈ 14× the ACP+ throughput). This stark difference is partly explained by the segment sizes used by BBR-d1m1, on an average about 14 KB, in comparison to the constant 1024 byte payload of an ACP+ packet. Recall our assumption that every new segment contains a fresh update. 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.

---

\(^{15}\)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 ACKs, which indicate losses and out-of-order packets received at the receiver. We observed very low percentages of duplicate ACKs (< 1%) for up to 20 nodes and up to ≈ 1.5% for up to 80 nodes. We matched the time of reception of dupACKs to retransmitted data to establish a correspondence. Even the retransmitted data percentages are very low (≈ 0.5%) for up to 80 nodes.

\(^{16}\)https://man7.org/linux/man-pages/man7/tcp.7.html
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 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. The much smaller 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, as we had observed in Section VIII.

5) 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 [11]) 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 \( RTT_{\text{base}} \) (see Figure 16a). This tells us that BBR-d1m1’s flow on an average saw an RTT of \( RTT_{\text{base}} \). It obtained a low throughput of 100 Mbps, which was an accidental consequence of the receiver buffer size settings of BBR-d1m1 that disallowed the congestion control algorithm to push bytes into the network at a larger rate. The higher throughput of BBR-d1m3 came with a higher age.

B. Age Over Shared and Contended Access

We compare the performance of the congestion control algorithms with ACP+ for when one or more TCP clients (sources) connect to a TCP server (monitor) on the cloud via a WiFi access point. The setup and methodology are same as in Section VIII. We use iPerf3 to generate TCP traffic from ORBIT nodes to the AWS server. We begin with the low contention configuration that has five or fewer nodes connect to the access point and later consider high contention.

1) Shared Network With Low Contention: The number of nodes connected to the access point are a maximum of 5. The WiFi link rate is set to 12 Mbps. Figure 22a 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.

The TCP algorithms, unlike ACP+, always fully utilize the bottleneck WiFi link. As is seen in Figure 22b, 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. Recall our results from Section VIII 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 22b, is far from saturating the 12 Mbps WiFi link.

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 XI-A) and also when WiFi access contention is low, as seen above. Figure 23a shows the age achieved by BBR for 10, 20, 40 and 80 clients connected to a fixed-rate WiFi access point, for rates 6, 12, and 24 Mbps. Observe 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 in comparison to the increase seen by sources using ACP+ (see Figure 13a, also shown as part of Figure 23a).

Figure 23a shows the BBR throughput per node. BBR has larger per node throughputs than ACP+ (see Figure 13c). However, the differences in throughputs are nominal for when there are 20 or more clients, at any choice of rate, and also for 10 clients at 6 Mbps. For fewer clients, ACP+ has much smaller throughputs than BBR. The constraining factor as regards optimization of age is the backhaul beyond the access (see Section VIII). The BBR throughputs that are similar to that of ACP+ are, however, obtained at a much larger RTT (not shown). As a result, BBR achieves a much larger age. The large RTT(s) when using BBR are 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.

XII. CONCLUSION

The control algorithm of ACP+ performs distributed congestion control with the goal of optimizing age. A traffic flow corresponding to an application that requires low age (an age flow) must coexist in the public Internet with other age flows and with traffic flows of other applications.

When multiple age flows share links in the Internet, flows using the ACP+ control algorithm must converge, from any network state, to end-to-end rates that result in an efficient and fair use of the network. The notions of efficiency and fairness are well accepted for TCP flows. In [56] efficiency is defined as operating the network close to its knee, where the knee is the point beyond which increase in throughput is small and is accompanied by a large increase in delay. Increases in
throughput beyond the knee lead to the cliff, beyond which packets are lost. Sharing is fair in case all flows sharing the same bottleneck link have equal throughputs.

There are no commonly accepted notions of efficiency and fairness in the context of multiple end-to-end age flows. Also, as we have demonstrated earlier, a shared bottleneck link may not be saturated by age flows and its use by the flows may not serve as an indicator of efficiency or fairness.

In our experiments we see that ACP+ achieves similar ages for age flows that share an access network and send updates to a monitor. Sharing is fair age-wise. However, as is evidenced by the recent works [57], [58] on fairness in end-to-end TCP congestion control, a wide range of network conditions including congestion in the core of the Internet, which has very high bandwidths and could have thousands of flows sharing it, and the impact of non-congestive delays [57] that could result in starvation of TCP flows, need careful consideration.

As applications with different requirements use the public Internet, different congestion control algorithms (CCA) will interact with each other. The typical approach so far has been for a CCA to be TCP-friendly [59], [60]. It has been known that TCP-friendliness may result in traffic flows becoming overly conservative in use of available bandwidth, which may be detrimental to applications. Also, newer CCA proposals like BBR have been shown to be TCP-unfriendly. In [61], a framework for congestion control based on congestion shares is proposed as an alternative to TCP-friendliness.

Our work [50] shows how ACP+ flows are starved by TCP flows sharing the same link, further demonstrating the need for congestion control frameworks that accommodate traffic flows with different requirements in the public Internet.

To conclude, we proposed the Age Control Protocol ACP+ (available at [62]) for applications that desire freshness of information at a destination (monitor). We evaluated ACP+ over a wide range of simulated networks and real-world end-to-end paths over the Internet. Our evaluation brings insights to the problem of age control over the Internet. We also compared and contrasted ACP+ with various state-of-the-art TCP congestion control algorithms. We showed why they are unsuitable for age control over the Internet.

ACKNOWLEDGMENT

The authors would like to thank the editor and reviewers for their feedback that helped in improving the paper.

REFERENCES

[1] Y. Sun, Y. Polyanskiy, and E. Uysal, “Sampling of the Wiener process for remote estimation over a channel with random delay,” IEEE Trans. Inf. Theory, vol. 66, no. 2, pp. 1118–1135, Feb. 2020.

[2] J. P. Champa, H. Al-Zubaidy, and J. Gross, “Statistical guarantee optimization for age of information for the D/G/1 queue,” in Proc. IEEE Conf. Comput. Commun. Workshops, Apr. 2018, pp. 130–135.

[3] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?” in Proc. IEEE INFOCOM, Mar. 2012, pp. 2731–2735.

[4] C. Kam, S. Kompella, and A. Ephremides, “Age of information under random updates,” in Proc. IEEE Int. Symp. Inf. Theory, Jul. 2013, pp. 66–70.

[5] C. Kam, S. Kompella, and A. Ephremides, “Effect of message transmission diversity on status age,” in Proc. IEEE Int. Symp. Inf. Theory, Jun. 2014, pp. 2411–2415.

[6] C. Kam, S. Kompella, G. D. Nguyen, and A. Ephremides, “Effect of message transmission path diversity on status age,” IEEE Trans. Inf. Theory, vol. 62, no. 3, pp. 1360–1374, Mar. 2016.

[7] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, “Age of information: An introduction and survey,” IEEE J. Sel. Areas Commun., vol. 39, no. 5, pp. 1183–1210, 2021.

[8] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, Fundamentals of Queuing Theory, vol. 399. Hoboken, NJ, USA: Wiley, 2018.

[9] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, “A survey of recent results in networked control systems,” Proc. IEEE, vol. 95, no. 1, pp. 138–162, Jan. 2007.

[10] T. Shreedhar, S. K. Kaul, and R. D. Yates, “ACP: Age control protocol for minimizing age of information over the Internet,” in Proc. ACM MobiCom, 2018, pp. 699–701, doi: 10.1145/3241539.3267740.

[11] T. Shreedhar, S. K. Kaul, and R. D. Yates, “An age control transport protocol for delivering fresh updates in the Internet-of-Things,” in Proc. IEEE 20th Int. Symp. (WoWMoM), Jun. 2019, pp. 1–7, doi: 10.1109/WoWMoM.2019.8793011.

[12] T. Shreedhar, S. K. Kaul, and R. D. Yates, “An empirical study of age in the cloud,” in Proc. IEEE INFOCOM - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), May 2021, pp. 1–6, doi: 10.1109/INFOCOMWKSHPS51825.2021.9484567.

[13] Network Working Group. TCP Congestion Control, document RFC 5681, 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681

[14] S. Ha, I. Rhee, and L. Xu, “CUBIC: A new TCP-friendly high-speed TCP variant,” ACM SIGOPS Operat. Syst. Rev., vol. 42, pp. 64–74, Jul. 2008.

[15] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, “TCP vegas: New techniques for congestion detection and avoidance,” in Proc. Conf. Comput. Commun. Archit. Protocols Appl., 1994, pp. 24–35.

[16] A. Baiocchi, A. P. Castellani, and F. Vacirca, “YeeAH-TCP: Yet another highspeed TCP,” Proc. PFLDnet, vol. 7, pp. 37–42, 2007.

[17] 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.

[18] M. Costa, M. Codreanu, and A. Ephremides, “Age of information with packet management,” in Proc. IEEE Int. Symp. Inf. Theory, Jun. 2014, pp. 1583–1587.

[19] L. Huang and E. Modiano, “Optimizing age-of-information in a multi-class queueing system,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2015, pp. 1681–1685.

[20] M. Costa, M. Codreanu, and A. Ephremides, “On the age of information in status update systems with packet management,” IEEE Trans. Inf. Theory, vol. 62, no. 4, pp. 1897–1910, Apr. 2016.

[21] 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,” IEEE Trans. Inf. Theory, vol. 65, no. 12, pp. 8305–8324, Dec. 2019.

[22] A. Soysal and S. Ulukus, “Age of information in G/G/1/1 systems: Age expressions, bounds, special cases, and optimization,” IEEE Trans. Inf. Theory, vol. 67, no. 11, pp. 7477–7489, Nov. 2021.

[23] E. Najm and E. Telatar, “Status updates in a multi-stream M/G/1/1 preemptive queue,” in Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Apr. 2018, pp. 124–129.

[24] R. D. Yates and S. K. Kaul, “The age of information: Real-time status updating by multiple sources,” IEEE Trans. Inf. Theory, vol. 65, no. 3, pp. 1807–1827, Mar. 2019.

[25] M. Moltanaf, M. Leinonen, and M. Codreanu, “On the age of information in multi-source queueing models,” IEEE Trans. Commun., vol. 68, no. 8, pp. 5003–5017, Aug. 2020.

[26] R. D. Yates, “Status updates through networks of parallel servers,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2018, pp. 2281–2285.

[27] C. Kam and S. Kompella, “On the age of information for non-preemptive queues in tandem,” Frontiers Commun. Netw., vol. 3, Nov. 2022, Art. no. 932111. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frcmn.2022.932111

[28] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, “Minimizing the age of information in broadcast wireless networks,” in Proc. 54th Annu. Allerton Conf. Commun., Control, Comput. (Allerton), Sep. 2016, pp. 844–851.

[29] Z. Jiang, B. Krishnamachari, X. Zheng, S. Zhou, and Z. Niu, “Decentralized status update for age-of-information optimization in wireless multiaccess channels,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2018, pp. 2276–2280.

[30] 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 Proc. PFLDnet, vol. 7, pp. 37–42, 2007.
A. M. Bedewy, Y. Sun, and N. B. Shroff, “Minimizing the age of information through queues,” IEEE Trans. Inf. Theory, vol. 65, no. 8, pp. 5215–5232, Aug. 2019.

A. M. Bedewy, Y. Sun, and N. B. Shroff, “The age of information in multihop networks,” IEEE/ACM Trans. Netw., vol. 27, no. 3, pp. 1248–1257, Jun. 2019.

Y. Sun, E. Uysal-Biyikoglu, and S. Kompella, “Age-optimal updates of multiple information flows,” in Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Apr. 2018, pp. 136–141.

A. Maatouk, Y. Sun, A. Ephremides, and M. Aissa, “Status updates with priorities: Lexicographic optimality,” in Proc. 18th Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw. (WiOPT), Jun. 2020, pp. 1–8.

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. Symp. Inf. Theory (ISIT), Jul. 2016, pp. 2569–2573.

R. Talak, S. Karaman, and E. Modiano, “Minimizing age-of-information in multi-hop wireless networks,” in Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton), Oct. 2017, pp. 486–493.

R. D. Yates, “Lazy is timely: Status updates by an energy harvesting source,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2015, pp. 3008–3012.

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 Trans. Inf. Theory, vol. 63, no. 11, pp. 7492–7508, Nov. 2017.

A. Arafa et al., “Sample, quantize, and encode: Timely estimation over noisy channels,” IEEE Trans. Commun., vol. 69, no. 10, pp. 6485–6499, Oct. 2021.

M. Klugel, M. H. Mandduhi, S. Hirche, and W. Kellere, “Aol-penalty minimization for networked control systems with packet loss,” in Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Apr. 2019, pp. 189–196.

J. Pan, A. M. Bedewy, Y. Sun, and N. B. Shroff, “Optimizing sampling for data freshness: Unreliable transmissions with random two-way delay,” in Proc. IEEE Conf. Comput. Commun. (INFOCOM), May 2022, pp. 1389–1398.

S. Kaul, M. Gruteser, V. Rai, and J. Kenney, “Minimizing age of information in vehicular networks,” in Proc. 8th Annu. IEEE Commun. Soc. Conf. Sens. Mesh Ad Hoc Commun. Netw., Jun. 2011, pp. 350–358.

E. T. Ceran, D. Gündüz, and A. György, “Average age of information with hybrid ARQ under a resource constraint,” IEEE Trans. Wireless Commun., vol. 18, no. 3, pp. 1900–1913, Mar. 2019.

E. Najm, R. Yates, and E. Soljanin, “Status updates through M/G/1/1 queues with HARQ,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2017, pp. 131–135.

C. Sönmez, S. Baghaee, A. Ergi¸si, and E. Uysal-Biyikoglu, “Age-of-information in practice: Status age measured over TCP/IP connections through WiFi, Ethernet and LTE,” in Proc. IEEE BlackSeaCom, Jun. 2018, pp. 1–5.

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

Tanya Shreedhar received the bachelor’s degree (Hons.) in engineering from Panjab University, Chandigarh, India, and the Ph.D. degree from the Indraprastha Institute of Information Technology, Delhi (IIIT-Delhi), India, in 2022. She was a Visiting Researcher with the Chair of Connected Mobility, Technical University of Munich, in 2020. She is currently a Post-Doctoral Researcher with the Department of Informatics, The University of Edinburgh. Her research interests include next-generation transport protocols, wireless and mobile systems, and Internet measurements.

Sanjit K. Kaul received the B.Tech. degree in electronics and communications engineering from the Birla Institute of Technology, Mesra, India, in 2000, and the Ph.D. degree in electrical and computer engineering from Rutgers University in 2011. He was with the telecommunications industry for four years after completing the bachelor’s degree. From 2005 to 2011, he was a Research Assistant with the Wireless Information Networks Laboratory (WINLAB), Rutgers University. He is currently a Professor with the Department of Electronics and Communications, Indraprastha Institute of Information Technology, Delhi (IIIT-Delhi). He works on problems in wireless systems research. His interests include age-of-information and networked autonomous systems.

Roy D. Yates (Fellow, IEEE) received the B.S.E. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 1983, and the S.M. and Ph.D. degrees in electrical engineering from MIT in 1986 and 1990, respectively. In 1990, he joined the Wireless Information Networks Laboratory (WINLAB), Department of Electrical and Computer Engineering, Rutgers University, where he was an Assistant Professor Emeritus in 2022. He is an author (with David Goodman) of the book titled Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers (John Wiley and Sons), currently in its third edition. He was a recipient of the 2003 IEEE Marconi Prize Paper Award in Wireless Communications, the Best Paper Award from the ICC 2006 Wireless Communications Symposium, and the 2011 Rutgers University Teacher-Scholar Award.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.