ACP: An End-to-End Transport Protocol for Delivering Fresh Updates in the Internet-of-Things

Tanya Shreedhar
Wireless Systems Lab, IIIT-Delhi
tanyas@iiitd.ac.in

Sanjit K. Kaul
Wireless Systems Lab, IIIT-Delhi
skkaul@iiitd.ac.in

Roy D. Yates
WINLAB, Rutgers University
ryates@winlab.rutgers.edu

Abstract—The next generation of networks must support billions of connected devices in the Internet-of-Things (IoT). To support IoT applications, sources sense and send their measurement updates over the Internet to a monitor (control station) for real-time monitoring and actuation. Ideally, these updates would be delivered at a high rate, only constrained by the sensing rate supported by the sources. However, given network constraints, such a rate may lead to delays in delivery of updates at the monitor that make the freshest update at the monitor unacceptably old for the application.

We propose a novel transport layer protocol, namely the Age Control Protocol (ACP), that enables timely delivery of such updates to monitors, in a network-transparent manner. ACP allows the source to adapt its rate of updates to dynamic network conditions such that the average age of the sensed information at the monitor is minimized. We detail the protocol and the proposed control algorithm. We demonstrate its efficacy using extensive simulations and real-world experiments. To exemplify, ACP achieves about 100 msec of reduction in age in comparison to a baseline that has sources send one update every round-trip-time (RTT). This is for multiple sources that send their updates over the Internet to monitors on another continent over an end-to-end link with round-trip-times of 185 msec.

I. 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 systems for applications such as health care, smart homes, transportation, and natural environment monitoring. 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. The updates are communicated over the Internet to a *monitor* that processes them and decides on any actuation that may be required.

For such applications, it is desirable that freshly sensed information is available at monitors. However, as we will see, simply generating and sending updates at a high rate over the Internet is detrimental to this goal. In fact, 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. In this work, we propose 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.
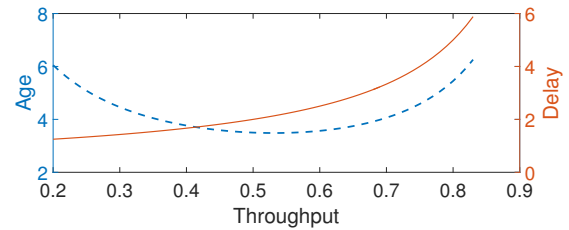

Figure 1: Interplay of the networking metrics of delay (solid line), throughput (normalized by service rate) and age. Shown for a M/M/1 queue [29] with service rate of 1. The age curve was generated using the analysis for a M/M/1 queue in [21].

This rate is such that the average age, where the age of an update is the time elapsed since its generation by the source, of sensed information at the monitor is kept to a minimum, given the network conditions. Based on feedback from the monitor, ACP adapts its suggested 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.

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. 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. 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 of application packets. As we show in Section III, 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.

Figure 1 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.

Many works, see Section II, have analyzed age as a Quality-of-Service metric for monitoring applications. Often such works have employed queue theoretic abstractions of networks. More recently in [28] the authors proposed a deep Q-learning based approach to optimize age over a given but unknown network topology. We believe our work is the first to investigate age control at the transport layer of the networking stack, that is over an end-to-end connection in an IP network and in a manner that is transparent to the application. A preliminary version of this work appeared as a poster [30]. Our specific contributions are listed next.

(a) We propose the Age Control Protocol, a novel transport layer protocol for real-time monitoring applications that wish to deliver fresh updates over IP networks. ACP regulates the rate at which a status updating source sends its updates to a monitor over its end-to-end connection in a manner that is application independent and makes the network transparent to the source. We argue that such a protocol, unlike other transport protocols like TCP and RTP, must have just the right number of update packets in transit at any given time.
(b) We demonstrate that TCP, which is the most commonly used transport protocol in the Internet is not suitable for the transport of update packets.
(c) We provide an extensive evaluation of the protocol using network simulations and real world experiments in which one or more sources sends packets to monitors. While our simulations are constrained to six hop paths between the sources and the monitors, in our real experiments we have sources send their updates over an inter-continental end-to-end IP connection. Over such a connection with a median round-trip-time of about 185 msec, ACP achieves a significant reduction in median age of about 100 msec ($\approx 33\%$ improvement) over age achieved by a protocol that sends one update every RTT.

The rest of the paper is organized as follows. In the next section, we describe related works. In Section III we demonstrate why the mechanisms of TCP are detrimental to minimizing age. In Section IV, we detail the Age Control Protocol, how it interfaces with a source and a monitor, and the protocol's timeline. In Section V we define the age control problem. In Section VI, we use simple queueing models to intuit a good age control protocol and discuss a few challenges.

Section IV details the control algorithm that is a part of ACP. This is followed by details on the evaluation methodology in Section VIII. We discuss simulation results in Section IX and results from real-world experiments in Section X. We conclude in Section XI.

II. RELATED WORK

The need for timely updates arises in many fields, including, for example, vehicular updating [20], real time databases [34], data warehousing [19], and web caching [7], [36].

For sources sending updates to monitors, there has been growing interest in the age of information (AoI) metric that was first analyzed for elementary queues in [21]. To evaluate AoI for a single source sending updates through a network cloud [15] or through an M/M/k server [16], [17], out-of-order packet delivery was the key analytical challenge. Packet deadlines are found to improve AoI in [18]. AoI in the presence of errors is evaluated in [6]. Distributional properties of the age process have also been analyzed for the D/G/1 queue under first-come-first-served (FCFS) [5], as well as single server FCFS and LCFS queues [10]. There have also been studies of energy-constrained updating [1], [2], [8], [26], [31], [35].

There has also been substantial efforts to evaluate and optimize age for multiple sources sharing a communication link [12], [14], [22], [24], [27]. In particular, near-optimal scheduling based on the Whittle index has been explored in [9], [13], [14]. When multiple sources employ wireless networks subject to interference constraints, AoI has been analyzed under a variety of link scheduling methods [23], [32]. AoI analysis for multihop networks has also received attention [33]. Notably, optimality properties of a Last Generated First Served (LGFS) service when updates arrive out of order are found in [3].

While the early work [20] 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. In this work, 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 [4], [25] 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.

III. 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
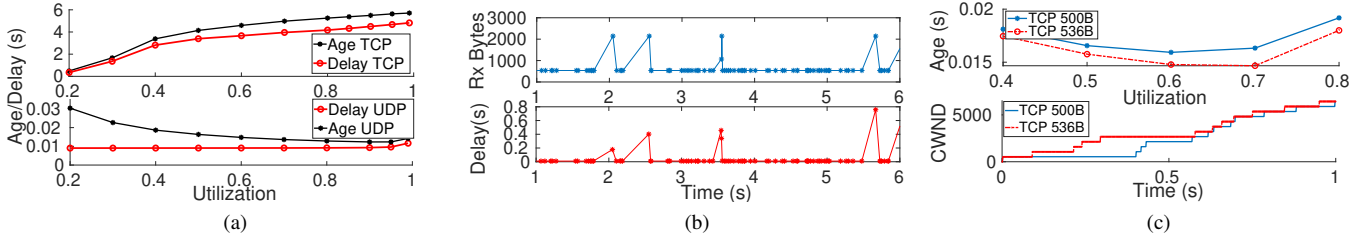
Figure 2: (a) Impact of packet errors on age and packet delay when using TCP and UDP (b) A time snapshot of delays suffered by update packets transmitted using TCP and the corresponding received bytes by the monitor. Packet error rate was set to 0.1 (c) Age as a function of packet size and how packet size impacts increase of the TCP congestion window.

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 2a illustrates the impact of packet error on TCP. A packet was dropped independently of other packets with probability 0.1. 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. 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 a utilization of ≈ 0.8 when UDP is used.

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 2b 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 2c 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 2c. 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 2c.

IV. 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 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. 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
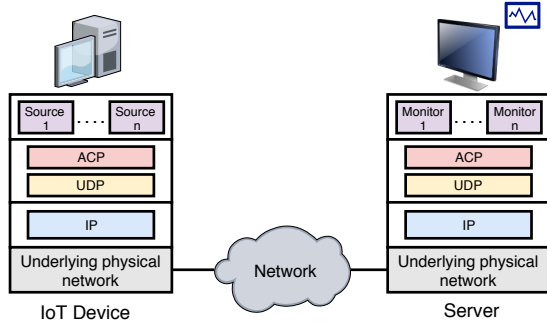
Figure 3: The ACP end-to-end connection.

Figure 5: Timeline of an ACP connection. The box **I** marks the beginning of the initialization phase of ACP. The boxed **C** denotes the ACP algorithm (Algorithm 1) executed when a new control epoch begins. The boxed **U** is executed when an ACK is received and updates $\overline{Z}, \overline{\mathrm{RTT}}$, and $\mathcal{T}$.

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 measurement from the source.

Figure 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.

V. THE AGE CONTROL PROBLEM

We will formally define the age of sensed information at a monitor. To simplify 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
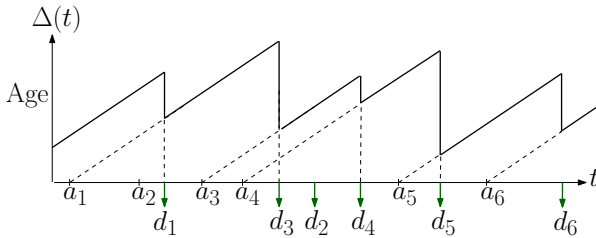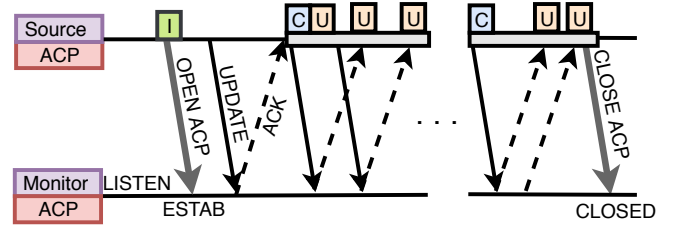
function of the age stochastic process is shown in Figure 4. The figure shows the timestamps $a_1, a_2, \ldots, a_6$ of 6 packets generated by the source. Packet i is received by the monitor at time d_i. At time d_i, packet i has age $d_i - a_i$. The age $\Delta(t)$ at the monitor increases linearly 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.
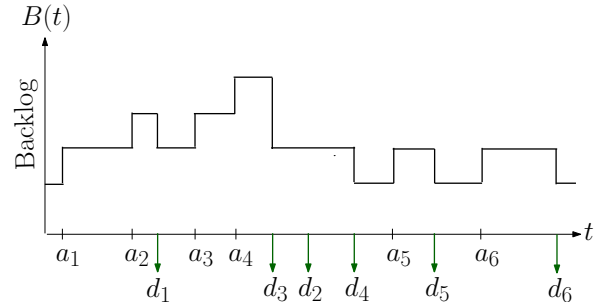

Figure 4: A sample function of the age $\Delta(t)$. Updates are indexed $1, 2, \ldots$. The timestamp of update i is a_i. The time at which update i is received by the monitor is d_i. Since update 2 is received out-of-sequence, it doesn't reset the age process.

Figure 6: 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 $d_2 > d_3$.
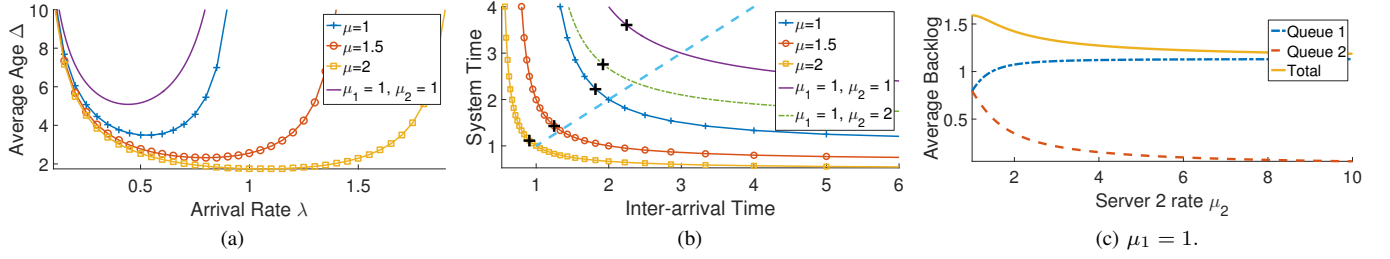
Figure 7: (a) Expected value of age as a function of update rate λ is shown for different queueing networks. (b) Average update packet system time is shown as a function of inter-arrival time $(1/\lambda)$. The green dashed line is the $45°$ line. The $+$ mark the age minimizing λ. (c) The backlog in each queue and the sum is shown as the service rate of the second queue increases from 1 to 5. Service rate of the first queue is $\mu_1 = 1$.

VI. 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 [29]. 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 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 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.

A. 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 [21]. 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 [29].

On the impact of non-stationarity and transient network conditions: Figure 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 every choice of λ. Getting these estimates, however, would require averaging over a large number of instantaneous age samples

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 I: Various P2P link configurations applied to the network diagram in Figure 11. 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.

Figure 8: Average backlogs at different nodes in the network, shown in Fig 11, at the optimal update rate. Net E is similar to Net A and not shown.

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 in transit. Figure 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 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, 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 increases, the system times converge to the average service time of a packet. For each queueing system, we also mark on its plot the inter-arrival time 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. However, for the two queues in tandem with service rates of 1 each, the inter-arrival time is a lot smaller than the system time. The implication being that on an average it is optimal to send slightly more than one (≈ 1.2) 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.

Note that these numbers are in fact the optimal (age minimizing) average number of packets in the system. Figure 8 shows how this optimal average backlog varies as a function of μ_2 for a given μ_1. The observations stay the same on
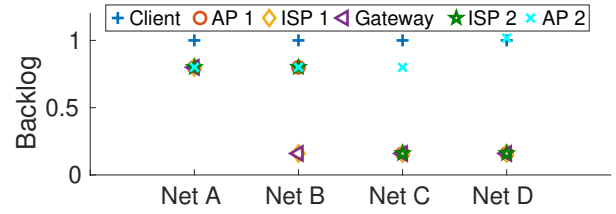
swapping μ_1 and μ_2. As μ_2 increases, that is as the second server becomes faster than the first ($\mu_1 = 1$), we see that the average backlog increases in queue 1 and reduces in queue 2, while the sum backlog gets closer to the optimal backlog for the single queue case. Specifically, as queue 1 becomes a larger bottleneck relative to queue 2, optimal λ must adapt to the bottleneck queue. The backlog in the faster queue is governed by the resulting choice of λ. For when the rates μ_1 and μ_2 are similar, they see similar backlogs. However, as is seen for when $\mu_1, \mu_2 = 1$, the backlog per queue is smaller than a network with only a single such queue. However, the sum backlog (≈ 1.6) is larger.

B. 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 11. 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 I. 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 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 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
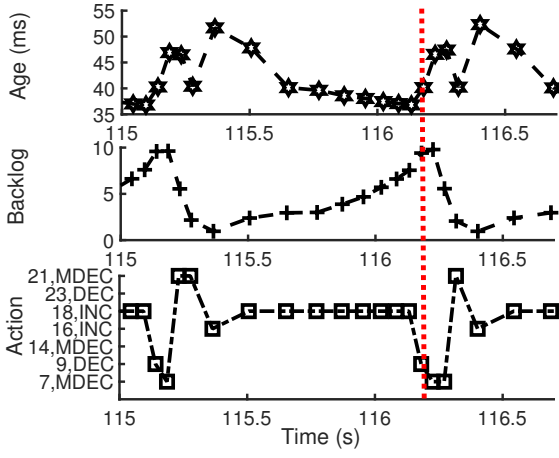
Figure 9: A snippet from the function of ACP. The y-axis of the plot showing actions denotes the action and the line number in Algorithm 1. Note the action marked by the dotted red line. At the time instant ACP observes an increase in both backlog and age and chooses (9,DEC) initially. However, there is still a significant jump in age. This results in the choice of multiplicative decrease (7,MDEC).

Algorithm 1 Control Algorithm of ACP

```
1: INPUT: b_k, δ_k, T̄
2: INIT: flag ← 0, γ ← 0
3: while true do
4:     if b_k > 0 && δ_k > 0 then
5:         if flag == 1 then
6:             γ = γ + 1
7:             MDEC(γ)
8:         else
9:             DEC
10:        flag ← 1
11:    else if b_k < 0 && δ_k > 0 then
12:        if flag == 1 && |b_k| < 0.5 * |b_k*| then
13:            γ = γ + 1
14:            MDEC(γ)
15:        else
16:            INC, flag ← 0, γ ← 0
17:    else if b_k > 0 && δ_k < 0 then
18:        INC, flag ← 0, γ ← 0
19:    else b_k < 0 && δ_k < 0
20:        if flag == 1 && γ > 0 then
21:            MDEC(γ)
22:        else
23:            DEC, flag ← 0, γ ← 0
24:    update λ_k
25:    wait T̄
```

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 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, which we do not demonstrate for lack of space, 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).

A good age control algorithm must not allow the end-to-end connection to drift into a high backlog state. As described next, ACP tracks changes in the number of backlogged packets and average age over short intervals, and in case backlog and age increase, ACP acts to rapidly reduce the backlog.

VII. THE ACP CONTROL ALGORITHM

Let the control epochs of ACP (Section V) 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 4), over the interval, at the monitor. It knows the time a_i a 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.

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 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 $t_{k+1} = t_k + \overline{T}$.
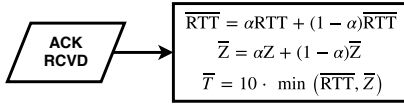
Figure 10: Update of $\overline{\text{RTT}}$, $\overline{Z}$, and $\overline{T}$, which takes place every time an ACK is received.

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 kth epoch that targets a change b_{k+1}^* in average backlog over an interval of length $\mathcal{T}$ with respect to the kth interval. The actions, may be broadly classified into (a) 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

$$\text{INC: } b_{k+1}^* = \kappa, \text{ DEC: } b_{k+1}^* = -\kappa,$$
$$\text{MDEC}(\gamma): b_{k+1}^* = -(1 - 2^{-\gamma})B_k, \quad (1)$$

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

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 1 summarizes how ACP chooses its action u_k as a function of b_k and δ_k. Figure 9 shows an example of ACP in action.

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 21) 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 18) ACP greedily attempts to increase backlog.
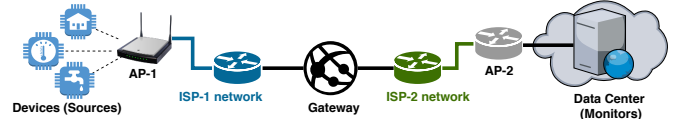

Figure 11: Sources are connected to the monitor via multiple routers and access points. Each source update travels over six hops. The first hop is between the source and access point AP-1. This could be either P2P or WiFi. The other hops that involve the ISP(s) and the Gateway are an abstraction of the 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 16) of backlog. If yes, and if the actual change in backlog was much smaller than the desired (line 12), 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.

VIII. EVALUATION METHODOLOGY

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

Figure 11 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. We simulated for no WiFi retries and a max retry limit of 7.

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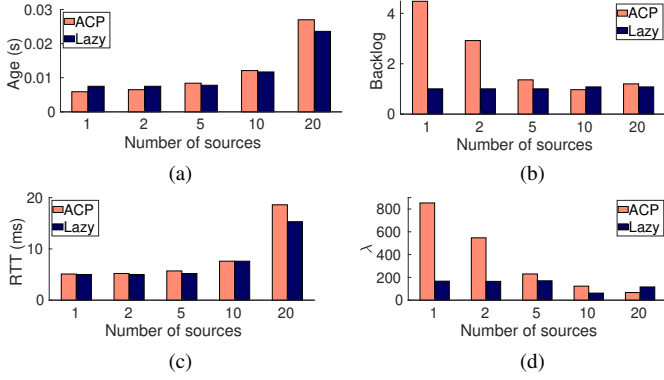
[1] https://www.nsnam.org/

Figure 12: (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 54 Mbps. The sources are spread over an area of 100 m^2. The standard deviation of shadowing was set to 4 dB.
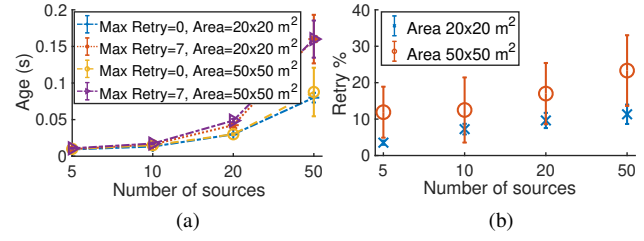

Figure 13: (a) Age and (b) retry rate as a function of number and density of sources and maximum retry limit. The vertical bars denote a region of ± 1 standard deviation around the mean (marked).
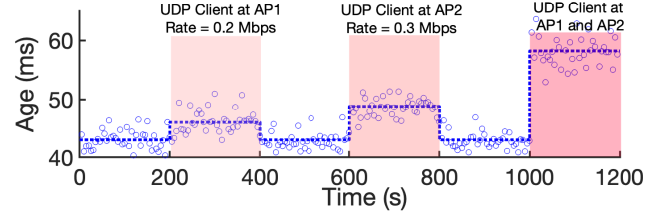

Figure 14: ACP adapts to network changes. Blue circles show the achieved age by an ACP client over time. A UDP client of rate 0.2 Mbps is connected to AP-1 at $200 - 400$ secs and $1000 - 1200$ secs. Another UDP client of rate 0.3 Mbps is connected to AP-2 at $600 - 800$ secs and $1000 - 1200$ secs. A darker shade of pink signifies a larger aggregate UDP load on the network.

with the YansWiFiPhyHelper[2]. Our simulated network is however limited in the number of hops, which is six.

We also evaluated ACP in the real-world by making $2 - 10$ sources connected to an enterprise WiFi access point, which is part of a university network, send their updates over the Internet to monitors that were running on a server with a global IP on another continent. This setup allows us to test ACP over a path with large RTT(s) and tens of hops. While the WiFi access point doesn't see much other traffic, 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.

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 hit and trial. 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 $\kappa \in \{1, 2\}$.

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

IX. SIMULATION RESULTS

Figure 12 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 (see Figure 12b) per source maintained by ACP is high. This is because, given the similar rate P2P links and higher rate WiFi

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

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. This results in ACP maintaining a backlog close to *Lazy* when the numbers of sources are 5 and more.

Figure 12d 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 the wireless access is shared by 20 sources. 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.

This artificial constraint of a very few hops combined with a single end-to-end path is removed in the real-world experiments that we present in the next section. As we will see,
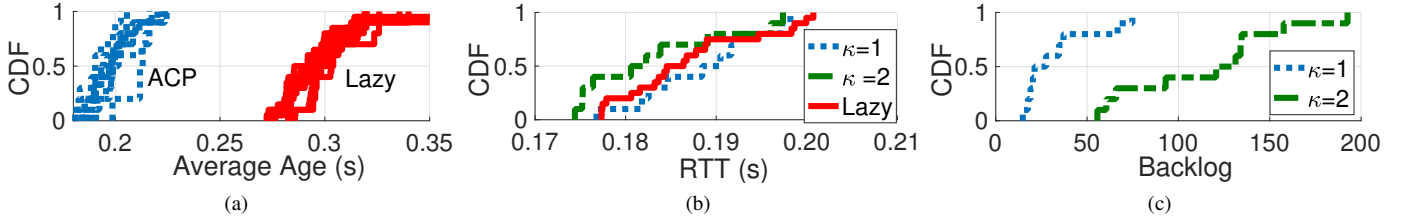
Figure 15: We compare the CDF(s) of average (a) Age (b) RTT and (c) Backlog obtained over 10 runs each of *Lazy* and ACP with step size choices of $\kappa = 1, 2$. The Age CDF(s) of all the 10 sources are shown.
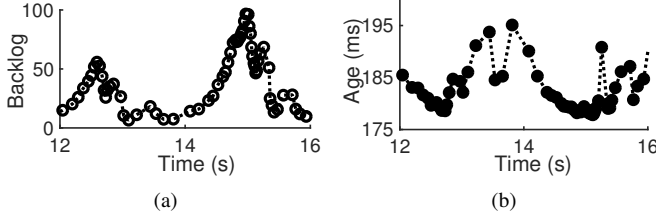

Figure 16: The time evolution of average backlog and age that resulted from one of the ACP source sending updates over the Internet.

sources accessing a common access point will maintain high backlogs over their end-to-end connections to their monitors.

The absolute improvements in average age achieved by ACP, see Figure 12a, 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 12c). 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 latter meant smaller λ of about 5 updates/second/source. The backlogs, as one would expect, were similar, however.

Next consider Figure 13a that shows the impact of maximum allowed retries, numbers of sources (varied from 5 to 50), and source density (areas of 50×50 m^2 and 20×20 m^2), on average age. The standard deviation of shadowing was set to 12. Note that age is similar for the two simulated areas for a given setting of maximum retries. However, it is significantly larger for when the max retry limit is set to 7 in comparison to when no retries are allowed. This is especially true when the network has larger numbers of sources. Larger numbers of sources witness higher rates of retries (Figure 13b, retry limit is 7) due to a higher rate of packet decoding errors that result from collisions over the WiFi medium access shared by all sources. Retries create a two-fold problem. One that a retry may keep a fresher update from being transmitted. Second, ACP, like TCP, confuses packet drops due to channel

errors to be network congestion. This causes it to unnecessarily reduce λ in response to packet errors, which increases age. In summary, retries at the wireless access are detrimental to keeping age low. Finally, observe in Figure 13a that the spread of ages achieved by sources is very small. In fact, we see that sources in a network achieve similar ages and in all our simulations the Jain's fairness index [11] was found to be close to the maximum of 1.

ACP adapts rather quickly to the introduction of other flows that congest the network. This is exemplified by Figure 14. We introduced one to two UDP flows at different points in the network used for simulation (Figure 11), where all links are 1 Mbps. ACP reduces λ appropriately and adapts backlog to desired levels.

X. INTER-CONTINENTAL UPDATES

We will show results for when 10 sources sent their updates to monitors on a server in another continent. 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, we observed that the number of hops was large, about 30, during these experiments.

Figure 15 summarizes the comparison of ACP and *Lazy*. Figure 15a 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 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 15b 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.

Lastly, consider a comparison of the CDF of average backlogs shown in Figure 15c. ACP exploits very well the fast end-to-end connection with multiple hops 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).

We end by showing snippets of ACP in action over the end-to-end path. Figures 16a and 16b show the time evolution of average backlog and average age, as calculated at control epochs. ACP increases backlog in small steps (see Figure 16a, 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.

XI. CONCLUSIONS

We proposed the Age Control Protocol, which is a novel transport layer protocol for real-time monitoring applications that desire the freshness of information communicated over the Internet. ACP works in an application-independent manner. It regulates the update rate of a source in a network-transparent manner. We detailed ACP's control algorithm that adapts the rate so that the age of the updates at the monitor is minimized. Via network simulations and real-world experiments, we showed that ACP adapts the source update rate well to make an effective use of network resources available to the end-to-end connection between a source and its monitor.

ACKNOWLEDGMENT

REFERENCES

[1] 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)*, pages 1–6, May 2018.

[2] 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.

[3] A. M. Bedewy, Y. Sun, and N. B. Shroff. Age-optimal information updates in multihop networks. *CoRR*, abs/1701.05711, 2017.

[4] 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)*, pages 1–6, April 2018.

[5] 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)*, pages 130–135, April 2018.

[6] K. Chen and L. Huang. Age-of-information in the presence of error. In *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, pages 2579–2584, 2016.

[7] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems (TODS)*, 28(4):390–426, 2003.

[8] 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)*, pages 112–117, April 2018.

[9] Y.-P. Hsu. Age of information: Whittle index for scheduling stochastic arrivals. In *isit*, pages 2634–2638, June 2018.

[10] 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*, abs/1804.06139, 2018.

[11] R. Jain, A. Durresi, and G. Babic. Throughput fairness index: An explanation. In *ATM Forum contribution*, volume 99, 1999.

[12] 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*, pages 2276–2280, June 2018.

[13] 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.

[14] 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)*, pages 844–851, Sept 2016.

[15] C. Kam, S. Kompella, and A. Ephremides. Age of information under random updates. In *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, pages 66–70, 2013.

[16] C. Kam, S. Kompella, and A. Ephremides. Effect of message transmission diversity on status age. In *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, pages 2411–2415, June 2014.

[17] C. Kam, S. Kompella, G. D. Nguyen, and A. Ephremides. Effect of message transmission path diversity on status age. *IEEE Trans. Info. Theory*, 62(3):1360–1374, Mar. 2016.

[18] 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)*, pages 2564–2568, 2016.

[19] 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*, IQIS '05, pages 28–39, Baltimore, Maryland, 2005. ACM. ACM ID: 1077509.

[20] 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.

[21] S. Kaul, R. Yates, and M. Gruteser. Real-time status: How often should one update? In *Proc. IEEE INFOCOM Mini Conference*, 2012.

[22] S. K. Kaul and R. Yates. Status updates over unreliable multiaccess channels. In *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, pages 331–335, June 2017.

[23] 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*, Mobihoc '18, pages 191–200, New York, NY, USA, 2018. ACM.

[24] 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)*, pages 124–129, April 2018.

[25] 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)*, pages 131–135, June 2017.

[26] 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)*, pages 1–5, July 2017.

[27] 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*, pages 1–6, Dec 2017.

[28] 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)*, pages 1–4, May 2018.

[29] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris. *Fundamentals of queueing theory*, volume 399. John Wiley & Sons, 2018.

[30] T. Shreedhar, S. K. Kaul, and R. D. Yates. Poster: Acp: Age control protocol for minimizing age of information over the internet. In *MOBICOM*, pages 699–701. ACM, 2018.

[31] 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*, 63(11):7492–7508, 2017.

[32] R. Talak, I. Kadota, S. Karaman, and E. Modiano. Scheduling policies for age minimization in wireless networks with unknown channel state. *CoRR*, abs/1805.06752, 2018.

[33] 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)*, pages 486–493, Oct 2017.

[34] M. Xiong and K. Ramamritham. Deriving deadlines and periods for real-time update transactions. In *The 20th IEEE Real-Time Systems Symposium, 1999. Proceedings*, pages 32–43. IEEE, 1999.

[35] R. Yates. Lazy is timely: Status updates by an energy harvesting source. In *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, 2015.

[36] H. Yu, L. Breslau, and S. Shenker. A scalable web cache consistency architecture. *SIGCOMM Comput. Commun. Rev.*, 29(4):163–174, Aug. 1999. ACM ID: 316219.