An Age Control 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*—Internet-of-Things (IoT) applications have 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 fresh, at a high rate constrained only by the supported sensing rate. However, such a rate may lead to network congestion related delays in delivery of updates at the monitor that make the freshest update at the monitor unacceptably old for the application. Alternately, at low rates, while updates arrive at the monitor with smaller delays, new updates arrive infrequently. Thus, both low and high rates may lead to an undesirably aged freshest update at the monitor.**

We propose a novel transport layer protocol, namely the Age Control Protocol (ACP), which enables timely delivery of such updates to monitors over the Internet in a network-transparent manner. ACP adapts the rate of updates from a source such that the average age of updates 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, including wireless access for the sources and an end-to-end connection with multiple hops to the monitor.

I. Introduction

Inexpensive IoT devices have 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, 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. See, for example, [7, Figure 3] for how age at the monitor varies as a function of source rate for simple first-come-first-served queues with memoryless arrival and service processes.

The requirement of freshness is not akin to requirements of other pervasive real-time applications like voice and video. While resilient to packet drops to a certain degree, they 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 and use the transmission control protocol (TCP). The congestion control algorithm of TCP, which optimizes the use of the network pipe for throughput, is detrimental to keeping age low. We detail the impact of TCP on age in [11, Section 3][1]. Its features of packet retransmissions and in-order delivery can keep fresh packets waiting at the monitor TCP for older packets to be successfully received. This causes large increases in age on transmission errors. Also, small sized updates may age more than larger ones as the congestion window size doesn't increase till a sender maximum segment size bytes are acknowledged. This delays the delivery of updates with fewer bytes.

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. In fact, while ACP chooses the best rate of sending updates, it uses UDP to transport them over the Internet.

Our specific contributions are listed next.

(a) We propose the Age Control Protocol (detailed in Sections II, III, IV, and V), a novel transport layer protocol for real-time monitoring applications that aims to deliver fresh updates over the Internet. ACP regulates the rate at which a 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.

978-1-7281-0270-2/19/$31.00

[1] We don't include evaluation of TCP in this paper because of limited space.

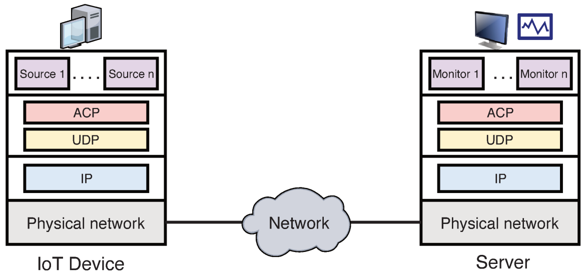


Figure 1: The ACP end-to-end connection.

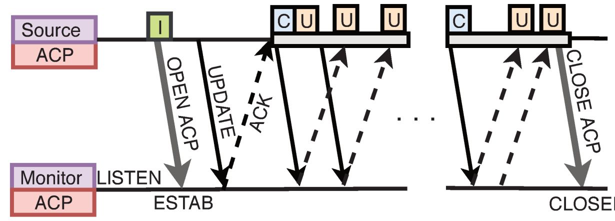


Figure 2: Timeline of an ACP connection. The box **I** marks the beginning of the initialization phase of ACP. The boxed **C** denotes the ACP algorithm (Algorithm 1) executed when a new control epoch begins. The boxed **U** is executed when an ACK is received and updates $\overline{Z}$, $\overline{\text{RTT}}$, and $\mathcal{T}$.

The goal is to keep the average age of sensed information at the monitor to a minimum, where the age of an update is the time elapsed since its generation by the source. Based on feedback from the monitor, ACP adapts rate to the perceived congestion in the Internet. Consequently, ACP also limits congestion that would otherwise be introduced by sources sending to their monitors at unnecessarily fast update rates. We argue that ACP, unlike other transport protocols like TCP and RTP, must maintain just the right number of update packets in transit at any given time.

(b) We provide an extensive evaluation of ACP (Sections VI, VII and VIII) using network simulations and real-world experiments in which one or more sources sends packets to monitors. To exemplify, over an inter-continental end-to-end IP connection with a median round-trip-time of about 185 msec, ACP achieves a significant reduction in the median age of about 100 msec ($\approx 33\%$ improvement) over age achieved by a protocol that sends one update every RTT.

We end this section with a brief on the related art. Prior works [5]–[7] have analyzed the metric of age of sensed information (AoI) for queue theoretic abstractions of networks. Works have considered optimizing age for multiple sources sharing a communication link, for example, [3], [4]. AoI has been analyzed under a variety of link scheduling methods [8], [13]. Multihop networks have also received attention [14]. Notably, optimality properties of a Last Generated First Served service when updates arrive out of order are found in [1].

Such AoI literature has 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. Closer to our goal, more recently, in [9] the authors proposed a deep Q-learning based approach to optimize age over a given but unknown network topology. A preliminary version of our work on ACP appeared as a poster [12].

II. 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 1 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 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 2 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.

III. 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

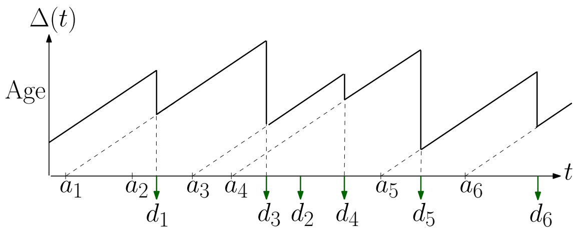


Figure 3: A sample function of the age $\Delta(t)$. Updates are indexed $1, 2, \ldots$. The timestamp of update i is a_i. The time at which update i is received by the monitor is d_i. Since update 2 is received out-of-sequence, it doesn't reset the age process.

the monitor up to time t. Recall that this is the time the update was generated by the source.

The age at the monitor is $\Delta(t) = t - z(t)$ of the freshest update available at the monitor at time t. An example sample function of the age stochastic process is shown in Figure 3. 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.

IV. 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 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 may age more if it ends up waiting for one or more 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 [10]. 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 other traffic. However, the resulting intuition is significant. Specifically, *a good age control algorithm must strive to have as many update packets of a source in transit as possible while simultaneously ensuring that these updates avoid waiting for other previously queued updates of the source*[2].

As described next, ACP tracks changes in the number of backlogged packets, which are updates for whom the source awaits an ACK from the monitor, and average age over short intervals. In case backlog and age increase, ACP acts to rapidly reduce the backlog.

V. The ACP Control Algorithm

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

Let $\overline{\Delta}_k$ be the estimate at the source ACP of the time average update age at the monitor at time t_k. This average is calculated over (t_{k-1}, t_k). To calculate it, the source ACP must construct its estimate of the age sample function (see Figure 3), 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

[2]Simulations that further build on this intuition may be found in [11]. We skip them here due to lack of space.

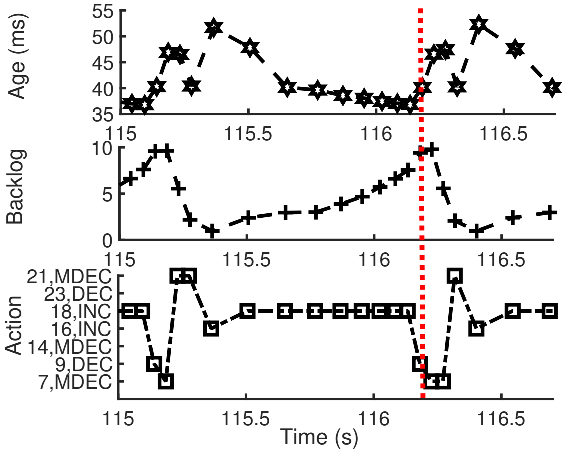


Figure 4: A snippet from the function of ACP. The y-axis of the plot showing actions denotes the action and the line number in Algorithm 1. Note the action marked by the dotted red line. At the time instant ACP observes an increase in both backlog and age and chooses (9,DEC) initially. However, there 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:** $b_k, \delta_k, \overline{T}$
2: **INIT:** $flag \leftarrow 0$, $\gamma \leftarrow 0$
3: **while** true **do**
4:   **if** $b_k > 0$ && $\delta_k > 0$ **then**
5:     **if** $flag == 1$ **then**
6:       $\gamma = \gamma + 1$
7:       MDEC(γ)
8:     **else**
9:       DEC
10:     $flag \leftarrow 1$
11:   **else if** $b_k < 0$ && $\delta_k > 0$ **then**
12:     **if** $flag == 1$ && $|b_k| < 0.5 * |b_k^*|$ **then**
13:       $\gamma = \gamma + 1$
14:       MDEC(γ)
15:     **else**
16:       INC, $flag \leftarrow 0$, $\gamma \leftarrow 0$
17:   **else if** $b_k > 0$ && $\delta_k < 0$ **then**
18:     INC, $flag \leftarrow 0$, $\gamma \leftarrow 0$
19:   **else** $b_k < 0$ && $\delta_k < 0$
20:     **if** $flag == 1$ && $\gamma > 0$ **then**
21:       MDEC(γ)
22:     **else**
23:       DEC, $flag \leftarrow 0$, $\gamma \leftarrow 0$
24:   update λ_k
25:   wait $\overline{T}$

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 $t_{k+1} = t_k + \overline{T}$. In all our evaluation we have used $\overline{T} = 10\mathcal{T}$. The resulting length of $\overline{T}$ was observed to be long enough to see desired changes in average backlog and age in response to a choice of source update rate at the beginning of an epoch. The source updates $\overline{\text{RTT}}$, $\overline{Z}$, and $\overline{T}$ every time an ACK is received.

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

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

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

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 4 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.

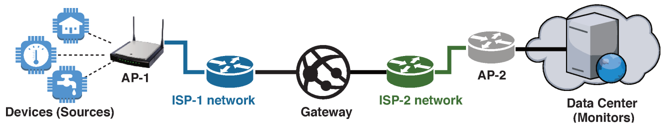


Figure 5: Sources are connected to the monitor via multiple routers and access points. Each source update travels over six hops. The first hop is between the source and access point AP-1. This could be either P2P or WiFi. The other hops that involve the ISP(s) and the Gateway are an abstraction of the 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.

VI. 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 5 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[3] together with the YansWiFiPhyHelper[4]. Our simulated network is however limited in the number of hops, which is six.

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

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

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.

VII. Simulation Results

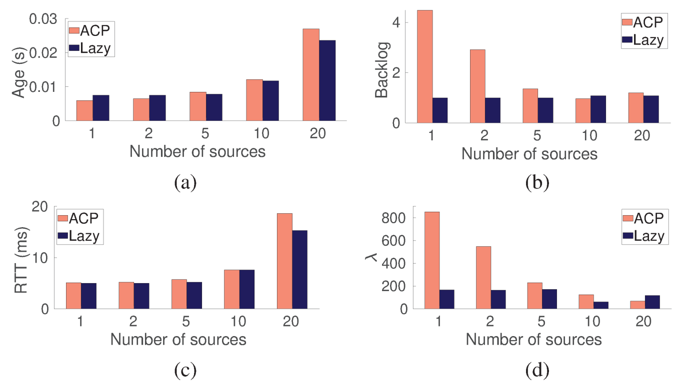


Figure 6: (a) Average source age (b) Average source backlog (c) Average source RTT and (d) Update rate λ for Lazy and ACP when all links other than wireless access are 6 Mbps. All sources used a WiFi PHY rate of 54 Mbps. The sources are spread over an area of 100 m^2. The standard deviation of shadowing was set to 4 dB.

Figure 6 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 6b) per source maintained by ACP is high. This is because, given the similar rate P2P links and higher rate WiFi link, when using ACP, the sources attempt to have their update packets in the queues of the access points and routers in the network. On the other hand, a source using *Lazy* sticks to sending just one packet every RTT on an average. Thus, the average backlog per source stays similar for different numbers of sources.

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

Figure 6d 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, 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 6a, 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 6c). 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.

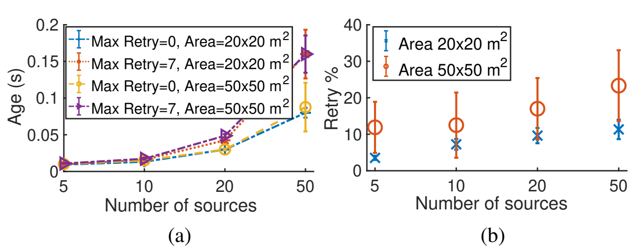


Figure 7: (a) Age and (b) retry rate as a function of number and density of sources and maximum retry limit. The vertical bars denote a region of ± 1 standard deviation around the mean (marked).

Next consider Figure 7a 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 7b, 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 7a 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 [2] was found to be close to the maximum of 1.

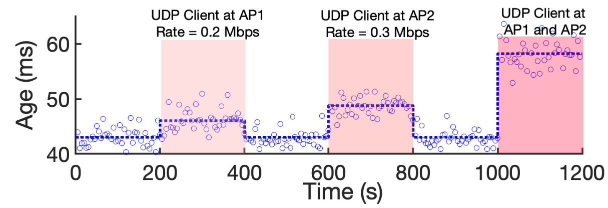


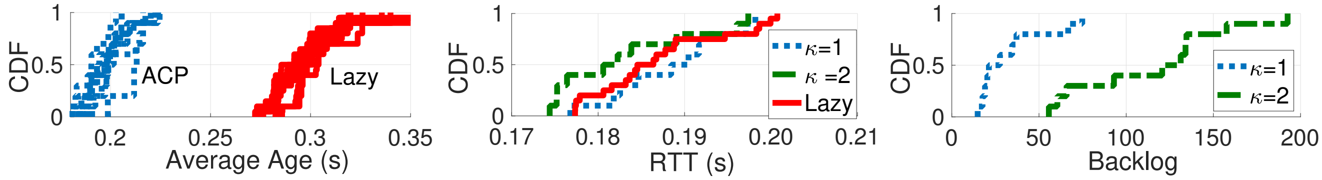
Figure 8: ACP adapts to network changes. Blue circles show the achieved age by an ACP client over time. A UDP client of rate 0.2 Mbps is connected to AP-1 at $200 - 400$ secs and $1000 - 1200$ secs. Another UDP client of rate 0.3 Mbps is connected to AP-2 at $600 - 800$ secs and $1000 - 1200$ secs. A darker shade of pink signifies a larger aggregate UDP load on the network.

ACP adapts rather quickly to the introduction of other flows that congest the network. This is exemplified by Figure 8. We introduced one to two UDP flows at different points in the network used for simulation (Figure 5), where all links are 1 Mbps. ACP reduces λ appropriately and adapts backlog to desired levels.

VIII. Inter-Continental Updates

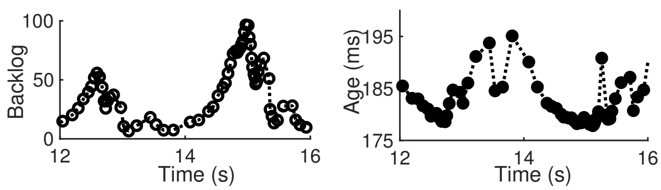
We will show results for when 10 sources sent their updates to monitors on a server in another continent. The sources, as described earlier, gained access to the Internet via an enterprise access point. The results were obtained by running ACP and *Lazy* alternately for 10 runs. Each run was restricted 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 9 summarizes the comparison of ACP and *Lazy*. Figure 9a 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,



(a) (b) (c)

Figure 9: 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.



(a) (b)

Figure 10: The time evolution of average backlog and age that resulted from one of the ACP source sending updates over the Internet.

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 9b 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 9c. 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 10a and 10b show the time evolution of average backlog and average age, as calculated at control epochs. ACP increases backlog in small steps (see Figure 10a, 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.

IX. 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

This research was funded by TCS Research Scholarship Program and Young Faculty Research Fellowship (Visvesvaraya Ph.D. scheme) from MeitY, Govt. of India.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

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

[11] T. Shreedhar, S. K. Kaul, and R. D. Yates. ACP: an end-to-end transport protocol for delivering fresh updates in the internet-of-things. *CoRR*, abs/1811.03353, 2018.

[12] 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.

[13] 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.

[14] 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.