An Emulation-Based Study of the *Age of Information*

Salman Qavi
sqavi@cs.stonybrook.edu

Many applications in today's world need near real-time information or status updates. For example, air traffic controllers and autonomous vehicles must continuously transmit location and position status updates from hundreds of sensors as soon as these are generated.

The faster such updates can be transmitted by the source and received by a monitor at the destination, the fresher the information will be and the more efficient use they can be made of. But how fast should we generate such updates to achieve the optimal results? In order to answer this, first we need to understand the difference between fresh and delayed data.

Age of Information (AoI), also known as *status age*, is a key performance indicator that measures the time that has elapsed since the generation of the packet that was most recently received at the destination [2]. In contrast, a packet *delay* measures the time interval between the generation of a packet and its delivery [2]. In other words, AoI allows us to quantify the freshness or staleness of data from the perspective of the destination [2].

In earlier works, [5] showed that there exists an *optimal rate* at which status updates need to be generated: making sensors send updates as soon as they are generated can create backlogs that lead the monitor at the destination to receive delayed status updates. On the other hand, reducing the rate at which updates are generated can result in receiving obsolete status updates [5]. The equation that defines AoI is shown in Equation 1 [6], where *u(t)* is the generation time (i.e. timestamp) of the newest status update the monitor has received by time *t*:

$$\Delta(t) = t - u(t) \tag{1}$$

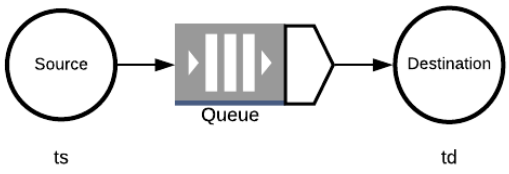

Figure 1: In our simple FCFS queuing implementation of a two nodes wireless network, the status updates are generated by the source at time ts and received by the monitor at time td.

Our goal is to set up a testbed and observe the behaviour of varying the sampling rates of Radio-Frequency (RF) status update generation against the *AoI*, finding out the sampling rate or the range at which RF status updates must be generated to achieve optimal results, and validating existing theories in literature with our work.

At the very outset, we planned to employ real transmitters and sensors for using RF data in our measurements. However, resource limitations due to COVID-19 restricted us to only use the emulator environment for our experiments. Due to prolonged lockdown, we were able to obtain a HackRF transmitter, a RTL-SDR dongle and an Odroid on April 27, 2020. Unfortunately, it turned out the transmitter transmits weak signals and the RTL-SDR sensor has connectivity issues. As a result, we opted out to using an emulated environment that closely resembles the actual testbed that we aimed to set up.

For this, we built an emulated testbed using Common Open Research Emulator (CORE) [1]. CORE is a network emulation framework that facilitates instantialization of lightweight virtual machines as *emulated nodes* within a wireless virtual network. This approach enabled us to replace actual transmitters and sensors with *emulated nodes* as the CORE network emulator can emulate wireless channel effects in real-time [1]. We used NRL's multi-generator (MGEN) network test tool to log status update generation and reception times for each traffic flow between the emulated nodes, and a Python script to trace through the MGEN log files and calculate the *AoI* as a function of time for different sampling rates.

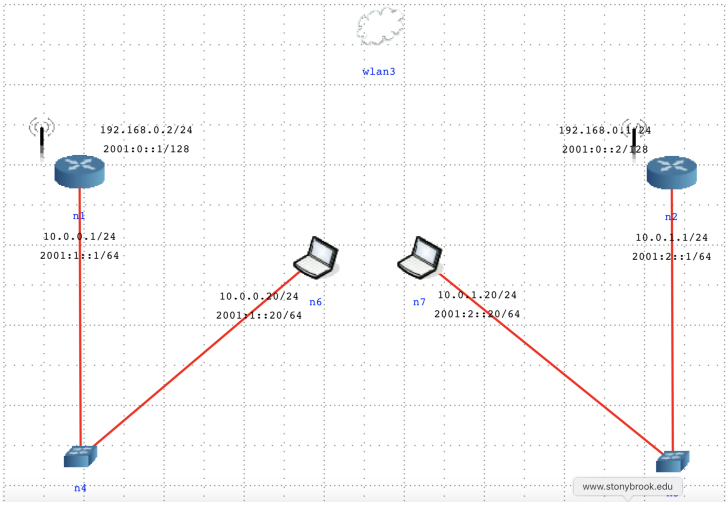

Figure 2: CORE GUI screenshot for two emulated nodes connected wirelessly

The emulation was executed on a MacBook Pro with a 2.4 GHz Intel Core i5 (Turbo Boost up to 4.1 GHz) CPU and 256 L2 and 6 MB L3 caches, 8GB of 2133 MHz LPDDR3 RAM and a 250.7 GB solid state disk. The laptop was running Mac OS Mojave (10.14.6). The CORE and EMANE environments were run on a virtual machine using VirtualBox 6.1 to run 64-bit Ubuntu 20.04 LTS. For the testbed, we used CORE 6.4, EMANE 1.2.6, MGEN 5.02 and Python 3.8.2.

We implemented a simple First-Come First-Serve (FCFS) queuing model consisting of two nodes as in Figure 1, where the status updates are generated at the source node s and are queued at the buffer before reaching the destination node d. MGEN script orig-

inates a *flow* of MGEN UDP destined for the monitor's address (IP address 192.168.0.2) port number 5000 beginning immediately when the script is executed. The status updates consist of 1024 byte packets and we keep the duration of all traffic flows to 30.0 seconds and maximum queue capacity to 10 packets. Packets are generated *periodically*. For each flow we keep the sampling rate (i.e. status update generation rate) constant and calculate the average *AoI* (i.e. differences between the generation and reception timestamps). For each sample rate, we take the *average AoI* over 30 iterations. We repeat this for varying sampling rates between 0 and 10 packets per second, and plot graphs for the *average AoI* against packets generated per second.

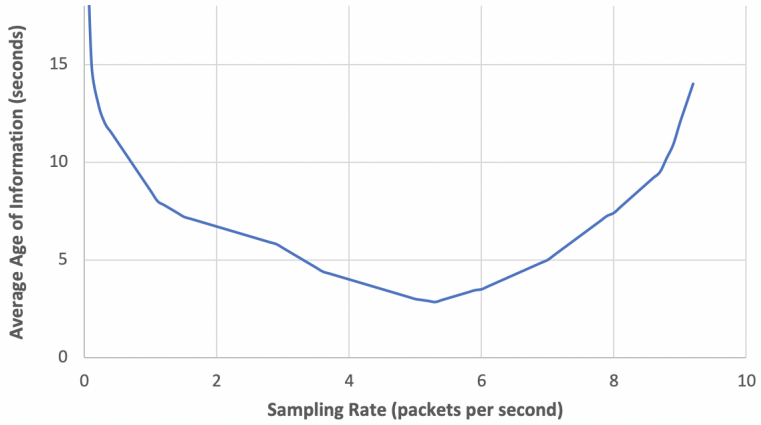

Figure 3: Average *AoI* decreases as more timely status updates reach the monitor until the rate is 5.5 packets per second. Beyond 6 packets per second, the queue starts to become increasingly saturated and the monitor starts receiving obsolete information by the time the packets get cleared off the queue and reach the monitor. As packet sampling rate approaches maximum queue size, the *AoI* increases exponentially due to packets getting dropped. The optimal sampling rate is reached between 5 and 6 packets per second.

The graph shown in Figure 3 reveals that when status updates are generated very rarely (i.e. sampling rate below 0.2 packets per second or 1 packet every 5 seconds), the *AoI* is maximum as updates comprise of mostly obsolete information. As we further increase the rate until a little over 5 packets per second, the *AoI* continues to decrease. This is because with increasing the frequency of status updates, the monitor now receives more timely updates (assuming all packets reach the monitor successfully and zero packet loss).

Increasing the sampling rate beyond this point, packets are being generated too frequently and results in increased *AoI*. This is because the queue is getting filled up and as more updates are generated, they are backlogged in the queue. By the time these updates are cleared from the queue they become stale and do not add value to the real-time communication system. Hence, beyond 6 packets per second, the *AoI* starts increasing again. As the rate approaches 10 packets per second, the *AoI* increases exponentially due to the fact that the queue is already full and increasing the sampling rate only results in packets being dropped, thereby increasing the *AoI* significantly. From this graph, we figure out that the optimal sampling rate is between 5 and 6 packets per second. This aligns with the findings from the literature as demonstrated in [3], [4]. [5], and [6].

The emulation may not have accounted for the effects of real systems such as processing time at the destination node and network interface queuing delays. In future work, we can take these into considerations and experiment for an optimal rate that reflect these effects. We can also use alternative scheduling approaches that are more efficient, e.g. Last-Come First-Serve (LCFS) mechanism, which would prioritize the ones arriving last and queue the ones arriving first. The packets arriving last carry the most recent updates and consequently the ones arriving early can be disregarded as they may not be adding any values to the communication system. Also, in a wireless network, distance between the nodes can have an effect on the optimal rate of sampling status updates. Additionally, relative movement of the transmitting and receiving nodes can alter the optimal rate. In future work, we can carry out experiments to validate these theories.

Acknowledgement

I would like to thank my advisor Professor Samir Das for his guidance and supervision and for providing me access to WINGS Lab and the resources for this project. I am grateful to Arani Bhattacharya for formulating the project idea and mentoring me from time to time as well as for providing me useful suggestions.

References

[1] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim. Core: A real-time network emulator. In *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pages 1–7, 2008.

[2] Igor Kadota, Elif Uysal-Biyikoglu, Rahul Singh, and Eytan Modiano. Minimizing the age of information in broadcast wireless networks. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 844–851. IEEE, 2016.

[3] C. Kam, S. Kompella, and A. Ephremides. Experimental evaluation of the age of information via emulation. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 1070–1075, 2015.

[4] C. Kam, S. Kompella, G. D. Nguyen, J. E. Wieselthier, and A. Ephremides. Controlling the age of information: Buffer size, deadline, and packet replacement. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 301–306, 2016.

[5] S. Kaul, R. Yates, and M. Gruteser. Real-time status: How often should one update? In *2012 Proceedings IEEE INFOCOM*, pages 2731–2735, 2012.

[6] C. Sönmez, S. Baghaee, A. Ergişi, and E. Uysal-Biyikoglu. Age-of-information in practice: Status age measured over tcp/ip connections through wifi, ethernet and lte. In *2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 1–5, 2018.