Benchmarking Drone Video Stream Latency In SteelEagle

Aditya Chanana, Mihir Bala, Thomas Eiszler, James Blakley,
Mahadev Satyanarayanan

May 2024
CMU-CS-24-128

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Low latency is critical for edge-enabled autonomous drones to do real-time computer vision tasks like target tracking effectively. We discuss our work on analyzing the latency of the real-time drone video stream using the SteelEagle autonomous drone system. By varying factors such as the cloudlet location, network type, and the video decoding library used, we show that the latency is bottlenecked not by the network but by the decoding of the RTSP video stream generated by the drone.

This material is based upon work supported by the U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W519TC-23-C-0003, and by the National Science Foundation under grant number CNS-2106862. The content of the information does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred. This work was done in the CMU Living Edge Lab, which is supported by Intel, Arm, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, IAI, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

Keywords: edge computing, drones, mobile networks, latency, bandwidth, edge-native applications, computer vision, machine learning

1 Introduction

SteelEagle is a software suite that transforms commercial-off-the-shelf (COTS) drones into fully autonomous, beyond-visual-line-of-sight (BVLOS) UAVs. It leverages edge computing to enable smaller and lighter drones while preserving the compute-intensive real-time sensing and autonomous capabilities of much larger drones [1]. SteelEagle transmits a real-time video stream from the drone's camera to a ground-based cloudlet where machine learning inference tasks, such as object detection, are offloaded. Based on the results, the drone is guided to avoid obstacles or track objects. There are latencies incurred in this process: (a) pre-processing on the drone to capture video frames from the camera and encode an H.264 stream (b) transmission of the video stream over the network to the cloudlet; (c) processing on the cloudlet; and (d) optional transmission of actuation commands back to the drone.

Obstacle detection and tracking are time-sensitive tasks that can be performed effectively only if the results of the offloaded computation are received promptly, ideally with very low latency. For example, a delayed decision to steer the drone to avoid an impending collision due to high latency can be catastrophic. This motivates us to analyze latency and identify where time is being spent. We try to tease through the contribution of the different factors by benchmarking different configurations, and changing one factor while keeping everything else the same. In this report, we show how our analysis led us to identify our choice of video decoding library as the main culprit leading to high latency in SteelEagle.

2 Background

In 2023, Bala et al [1] introduced the SteelEagle system to achieve autonomy for inexpensive flight platforms using edge computing. SteelEagle can perform real-time computer vision tasks including detection and tracking of targets with a drone platform weighing less than 350g and costing less than $800. We build upon Bala et al's work by quantifying the contribution of different factors that contribute to the end-to-end latency.

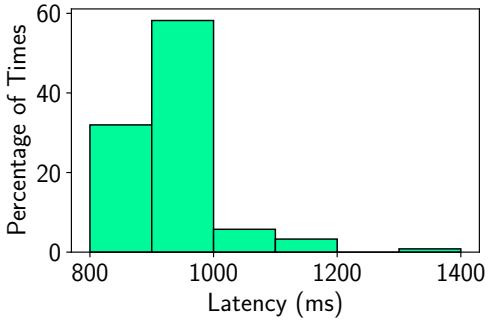


Figure 1: Original SteelEagle Latency [1]

Bala et al reported a mean drone-to-cloudlet latency of 933 ms (Figure 1). Such a high latency distribution makes it challenging to effectively perform computer vision inference tasks such as object detection and tracking. A round-trip latency from drone to cloudlet and back of over a second results in low drone agility. The drone is likely to lose track of fast-moving subjects or be forced to fly at lower speeds to circumvent obstacles in time.

3 Setup

We replicate the setup used by Bala et al in SteelEagle [1]. We use the Parrot Anafi drone (Figure 2) [4], which transmits a 720p H.264 RTSP stream at 30 FPS over UDP. This video stream is not configurable as it is generated by the hardware on the drone; both the frame rate and resolution cannot be changed.

Figure 2: Parrot Anafi

To reduce the impact of adverse network conditions, it uses a slice encoding and intra-refresh scheme that disperses keyframe slices across multiple transmission packets [5]. Software decoding of the H.264 compressed video stream is required to obtain individual video frames.

The Parrot Anafi drone lacks cellular connectivity, which restricts its offloading ability. The Onion Omega 2 LTE (Figure 3), a single-board computer that supports both Wi-Fi and 4G LTE, is attached to the drone as a payload and used as a communications hub to mitigate this. The drone is connected to the Onion Omega over Wi-Fi and a VPN tunnel is set up over LTE between the Onion and the cloudlet, allowing the cloudlet to communicate with the drone. We use the T-Mobile commercial LTE network unless stated otherwise. The drone exposes an API called Parrot Ground SDK that gives full flight control and access to on-board sensors.

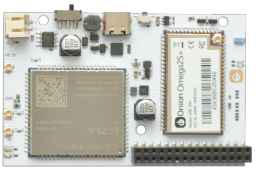


Figure 3: Onion Omega

4 Measuring latency

Round-trip latency, from drone to cloudlet and back, defines the drone's agility as it determines how fast the drone can respond to environmental changes. The Anafi, being a commercial product that restricts modification of its onboard software, effectively functions as a black box. Unfortunately, this black box nature makes it impossible to benchmark the round-trip latency. Even calculating the one-way latency is challenging since it is not possible to add instrumentation on the drone that records timestamps of when each frame is sent.

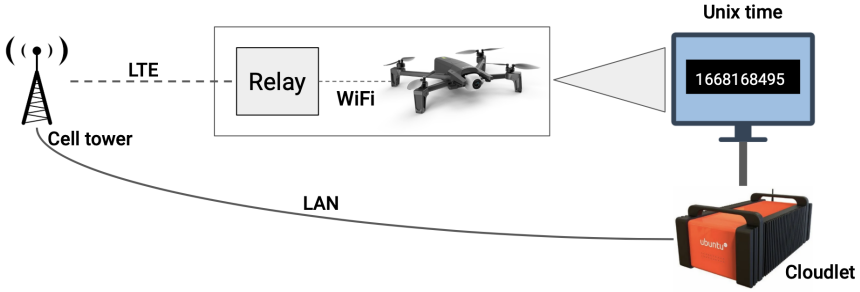


Figure 4: Latency Measurement Technique

To get around these limitations, we use the technique used by Bala et al for measuring the one-way drone-to-cloudlet video stream latency. The drone is kept stationary in a lab setting with its camera pointing at a display connected to the cloudlet showing the current time at millisecond granularity (Figure 4). The drone captures images of the timestamp displayed and transmits them to the cloudlet through the SteelEagle pipeline. The cloudlet records the timestamp at which each frame is received, storing the frame as a file along with this recorded timestamp. In post-processing, the files are manually processed to compute the difference between the timestamps to get the total end-to-end time. It can be more practical to use a display connected to a local computer that is time synchronized to the same NTP server as the cloudlet, instead of connecting a display to the cloudlet directly. This is sometimes the only way, for example, if the cloudlet is an EC2 VM running on AWS.

There are many components to the latency measured using this technique, some making up a bigger proportion than others. First, there is a delay before the timestamp appears on the display that depends on the refresh rate of the display monitor used. Second, there is a delay incurred in capturing the timestamp by the drone's camera, encoding the frame into a video stream, and transmitting it over WiFi. Third, the Onion relay adds delays as it receives packets over WiFi and forwards them via its LTE modem. Fourth, there is network latency over the LTE network as the packets go through the internet backbone before finally ending up at the cloudlet. Finally, the SteelEagle system adds latency as it receives enough packets for a single frame and then decodes and processes them.

5 Measurements

Table 1: Drone-to-Cloudlet Latency over T-Mobile IoT (in ms)

Configuration	Average	Median	p95	Min	Max
Cloudlet					
FFmpeg	888 ± 30	887	917	838	1,030
PDrAW	380 ± 15	379	402	344	420
AWS Small					
FFmpeg	536 ± 75	521	600	473	936
PDrAW	429 ± 21	427	467	387	475
AWS Big					
FFmpeg	870 ± 20	868	900	837	925
PDrAW	367 ± 20	365	392	327	416

50 samples obtained for each configuration.

Table 1 summarizes the measurements collected as part of our work. We begin by explaining the various configurations used and then analyze how the measurements change with each configuration to identify potential bottlenecks.

5.1 Configurations

The configurations used in our experiments in Table 1 vary by the location of the SteelEagle backend and the library used for decoding the drone video stream. All configurations use T-Mobile's commercial LTE network for IoT devices.

Location of backend. Three backend locations with different specifications, each offering varying levels of performance, are considered to evaluate how hardware scale-up affects system latency. The SteelEagle backend is either set up on a bare-metal server on CMU's campus, labeled "Cloudlet", or on an EC2 VM in AWS East (Virginia). Two types of EC2 instances are used, "AWS Small" and "AWS Big". The CMU Cloudlet has two Intel® Xeon® E5-2699 v3 CPUs clocked at 2.30 GHz for a total of 72 vCPUs, 128GB main memory, and an NVIDIA® GeForce® GTX 1080 Ti GPU. AWS denotes a `g4dn.xlarge` EC2 instance which has 4 vCPUs, 16GB of memory, and an NVIDIA T4 GPU. AWS Big is a `g4dn.16xlarge` EC2 instance which has 64 vCPUs, 256GB main memory, and also an NVIDIA T4 GPU.

Video decoding library. Two video decoding libraries are considered: FFmpeg and PDrAW (pronounced "pedro"). FFmpeg [2] is an open-source project that offers libraries for video encoding/decoding and multiplexing/demultiplexing. FFmpeg is known for forming a core part of the VLC media player. We interact with FFmpeg through OpenCV [3], which offers the ability to use FFmpeg as a backend for its video capture APIs.

PDrAW [6] is a part of Parrot's Ground SDK software. Similar to FFmpeg, it includes multiplexing/demultiplexing abilities and can read from the RTSP stream generated by the Parrot Anafi drone. Unlike FFmpeg, however, PDrAW is intended as a video player for RTSP and MP4 videos and lacks general-purpose encoding and decoding abilities.

5.2 Analysis

The SteelEagle system was initially configured to use FFmpeg for decoding the real-time video stream, using it as a backend for OpenCV video capture. The latency measurements obtained by Bala et al [1] also

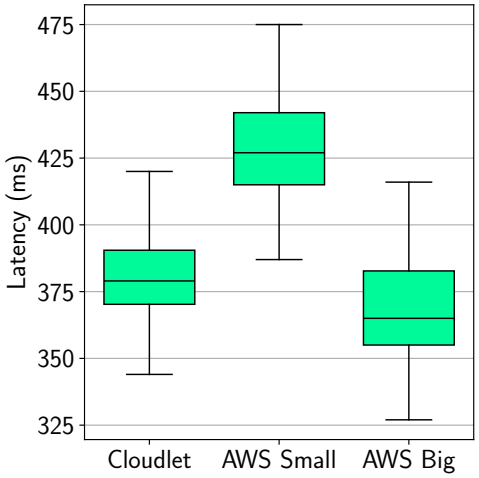


Figure 6: Latency Using PDrAW

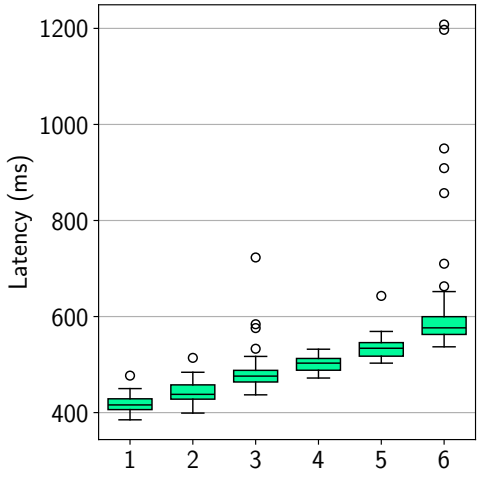


Figure 7: Latency vs. Number of FFmpeg Threads

Each box extends from the first quartile (Q_1) to the third quartile (Q_3), with a line at the median. Whiskers extend from the box to the farthest data point lying within 1.5x the inter-quartile range ($IQR = Q_3 - Q_1$) from the box. Circles represent outliers. 50 samples obtained for each configuration.

used FFmpeg with the CMU cloudlet as a SteelEagle backend (Figure 1). However, the data obtained in our experiments does not include inference time.

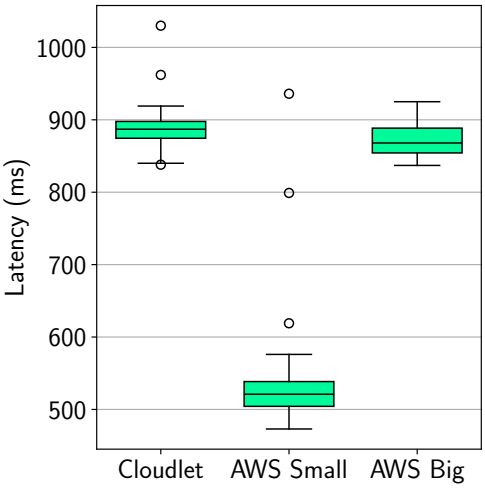
Our analysis of the SteelEagle video stream latency was performed by switching to a different SteelEagle configuration with only one variable being changed, either the backend location or the video stream decoding library used. We present the configuration changes in chronological order in which we performed them and discuss how the resulting findings eventually led us to identify scale-out issues with FFmpeg as a major contributor to high latency.

5.2.1 CMU Cloudlet to AWS Small

As shown in Figure 5, moving from the CMU Cloudlet to AWS Small for the SteelEagle backend leads to an extreme drop in latency, going down from p95 latency of 917 ms to 600 ms (see Table 1). That amounts to a 1.5× speedup, with an improvement of over 300 milliseconds.

Discussion. Initially, we attributed this improvement to network optimizations by T-Mobile. In early 2024, T-Mobile optimized its network to create a local breakout to the AWS East servers in Virginia. With this local breakout, packets heading for the AWS data center in Virginia would obtain direct routing to Virginia leading to fewer network hops than if they had to travel the entire internet backbone. This optimization allows for lower latency to an EC2 VM running in AWS than to a server running on CMU's campus.

However, our measurement of the round-trip network latency using the `ping` tool found the latency to AWS via T-Mobile to be 38 ms instead of 50 ms to the CMU Cloudlet on average. Hence, this latency reduction of 12 ms on average was not sufficient to explain the 300 ms drop in latency moving to AWS.



Each box extends from the first quartile (Q_1) to the third quartile (Q_3), with a line at the median. Whiskers extend from the box to the farthest data point lying within 1.5x the inter-quartile range ($IQR = Q_3 - Q_1$) from the box. Circles represent outliers. 50 samples obtained for each configuration.

Figure 5: Latency with FFmpeg

5.2.2 FFmpeg to PDrAW

Replacing FFmpeg with PDrAW for video decoding yields a significant improvement for AWS Small, with a reduction in p95 latency from 560 ms to 467 ms. Surprisingly, p95 latency with the CMU Cloudlet went down from 917 ms to 402 ms, a speedup of almost 2.3×.

Discussion. CMU Cloudlet has much lower latency than AWS Small when using PDrAW for video decoding, but much higher latency when using FFmpeg. This discrepancy is unexpected, as one would anticipate CMU Cloudlet to perform consistently better or worse across both configurations. Given that AWS Small has fewer logical CPUs than CMU Cloudlet, we extended our evaluation to a more powerful EC2 VM, AWS Big, to investigate whether it can offer even lower latency than AWS Small. The primary question now is: can AWS Big outperform both AWS Small and CMU Cloudlet in terms of latency with PDrAW? Additionally, what factors are contributing to the latency variations observed in the CMU Cloudlet? These questions drive our subsequent analysis, aiming to understand the underlying causes of these performance discrepancies and to identify optimal configurations for video decoding.

5.2.3 AWS Small to AWS Big

Surprisingly, AWS Big performed similarly to the CMU Cloudlet with p95 latencies around 900 ms for both. This is a huge degradation over using AWS Small, a weaker EC2 VM, which was found to have a p95 latency of 467 milliseconds.

Discussion. CMU Cloudlet has 72 logical CPUs, while AWS Big has 64, providing a similar computational capacity. In contrast, AWS Small has only 4 logical CPUs. This led to the hypothesis that FFmpeg's performance degrades with increased parallelism, struggling to scale effectively with higher CPU counts. Our measurements support this hypothesis: CMU Cloudlet, with 72 CPUs, exhibited the highest latency using PDrAW, with a mean of 887 ms. AWS Big, with 64 CPUs, showed a slightly lower mean latency of 870 ms, while AWS Small, with only 4 CPUs, achieved the lowest mean latency of 536 ms. To verify this hypothesis, we next consider a way to modify the number of CPUs used for FFmpeg on the same hardware.

5.2.4 Configure FFmpeg to Use Fewer Threads

Table 2: Multi-threaded FFmpeg Latency (ms)

Threads	Average	Median	p95	Min	Max
1	417 ± 19	416	447	385	477
2	442 ± 22	438	480	399	514
3	485 ± 45	476	563	437	723
4	502 ± 16	503	531	472	532
5	535 ± 24	534	566	503	643
6	625 ± 146	576	932	537	1,208

50 samples obtained for each row.

OpenCV provides the option CAP_PROP_N_THREADS to set the number of threads used for the FFmpeg backend. To test the hypothesis presented in Section 5.2.3, measurements were obtained by varying the value of this option from 1 to 6 (see Table 2 & Figure 7).

Discussion. FFmpeg has the best performance when using just a single thread. As we increase the number of threads used we observe negative scale-out characteristics and performance degrades substantially. These results provide conclusive evidence that the use of multi-threaded FFmpeg in SteelEagle has contributed to high system latency.

FFmpeg's negative scale-out characteristics may stem from its optimization for throughput rather than latency. Experiments have confirmed that increasing the number of threads enhances throughput but adversely affects latency. One possible reason is that multi-threaded FFmpeg might buffer frames longer than necessary, with each thread potentially maintaining its own frame buffer. Additionally, adding more threads can increase contention if they access the same data concurrently. Overhead from frequent thread switching can occur if spinlocks are not used appropriately. Cache line contention can lead to unnecessary cache invalidation when threads share cache lines. Furthermore, the decoding task may not be inherently parallelizable, so increasing the number of threads only adds unnecessary overhead. FFmpeg, being a general-purpose library, may be struggling with the specific compression scheme the Anafi uses. Understanding the true reason for this behavior will require detailed code analysis.

5.3 Contribution of Network Latency

Table 3: Latency (in ms) For Drone Connected To CMU Cloudlet Over WiFi

Configuration	Average	Median	p95	Min	Max
WiFi + FFmpeg	842 ± 17	841	868	807	879
WiFi + PDrAW	341 ± 22	338	374	298	419

50 samples obtained for each configuration.

While we achieved a significant reduction in end-to-end latency by switching to PDrAW, a mean of 380 ms is still high. We now aim to understand the LTE network's contribution in our search for the next bottleneck. While we have obtained an ad-hoc estimation using the `ping` tool in Section 5.2.1, we seek to more concretely quantify it. This is motivated by the concern that `ping` ICMP packets might receive preferential treatment compared to UDP packets.

To analyze the contribution of the LTE network latency, the experimental setup was modified such that the Anafi drone is now connected directly to the CMU Cloudlet over WiFi, eliminating the Onion relay. As before, the drone captures frames containing timestamps shown on a display that are sent directly to the backend, this time directly over WiFi. Table 3 contains the results from this experiment. The p95 latency for PDrAW over WiFi is 374 ms. Table 1 shows that the p95 latency with PDrAW over T-Mobile IoT is 402 ms. The hop over T-Mobile only added 28 ms to the p95 latency.

This suggests that the cellular network is currently not the bottleneck for SteelEagle. There are certain limitations to this claim, as the use of a USB WiFi dongle might have added considerable latency to the results. This would mean that perhaps the numbers obtained in Table 3 are higher than they should be, leading us to believe that the network latency is much less than actual.

5.4 CBRS-based Private LTE Network

To further validate that the network latency is not the bottleneck, we evaluate the system using CMU's CBRS-based private LTE network. Table 4 shows the measurements obtained. Instead of the Onion relay, a CBRS cellular dongle connected to a computer was used as a relay because the Onion Omega's LTE modem does not support CBRS frequencies. The CBRS network shows similar benefits as seen before when switching from FFmpeg to PDrAW.

CBRS results are worse across the board compared to T-Mobile IoT in these configurations. Using the CBRS network requires no more than 2-3 network hops to reach the CMU Cloudlet, which is much lower than T-Mobile. Yet, we see a p95 latency that is higher by 24 milliseconds than that obtained when reaching the CMU Cloudlet via T-Mobile IoT when using PDrAW, and higher by 205 milliseconds when using FFmpeg (compare with Table 1).

Table 4: Latency (in ms) For CBRS-Based Private LTE Network

Configuration	Average	Median	p95	Min	Max
CMU Cloudlet					
CBRS + FFmpeg	742 ± 139	720	764	691	1,696
CBRS + PDrAW	459 ± 27	456	504	396	520
AWS					
CBRS + FFmpeg	766 ± 34	757	807	724	927
CBRS + PDrAW	504 ± 25	503	548	448	564

50 samples obtained for each configuration.

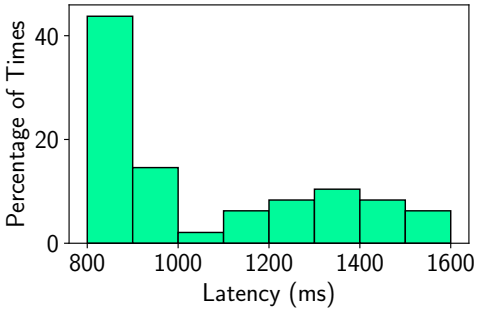


Figure 8: Latency With Phone & T-Mobile

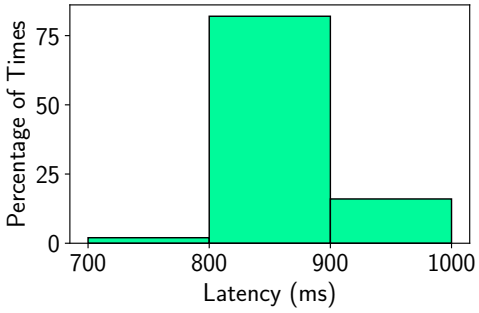


Figure 9: Latency With Phone & CBRS

To alleviate concerns about the overhead added by the use of the CBRS cellular dongle, we evaluate the system using the Google Pixel phone as a relay, which can connect to both the T-Mobile 4G LTE network and CMU's CBRS-based private LTE network. Figure 8 and Figure 9 show measurements obtained by using this relay for both networks using the CMU Cloudlet as the backend. When using T-Mobile for the network we see a very heavy-tailed distribution which is not the case when using the CBRS-based private LTE network. CBRS brings a latency advantage when used with the Google Pixel phone as a relay. This is what we expect since the CBRS pipeline involves fewer network hops.

Further research is required to understand why these benefits are not observed when using the CBRS cellular USB dongle as a relay. The CBRS cellular dongle used in these experiments is of questionable quality and may be adding significant latency overhead. Additionally, there may be other ways to optimize the private network to better support the SteelEagle workload.

6 Future Work

The version of FFmpeg used in this work does not utilize GPU hardware acceleration. Compiling FFmpeg to use NVIDIA's NVENC/NVDEC video encoding and decoding hardware support could potentially make it perform better. Can hardware-accelerated FFmpeg achieve lower latency than PDrAW? Conventional wisdom says that workloads that are inherently parallelizable benefit the most from the use of GPUs. Other types of workloads may lead to higher perceived latency due to resource underutilization. Can video decoding effectively leverage the GPU?

Profiling the video decoding pipeline on the SteelEagle backend, at a more granular level, will reveal how much of the latency it precisely contributes. There might be potential to further optimize the code. It could answer why FFmpeg does worse when the number of threads used is increased.

Lastly, more experimentation can be done with a Google Pixel smartphone as the relay. Unlike the Onion Omega, a Google Pixel is protected from the elements, has on-board compute, and supports both CBRS and LTE cellular. Can a Google Pixel smartphone be as performant as the Onion Omega?

7 Conclusion

SteelEagle is currently bottlenecked by the decoding of the RTSP video stream, not network latency. We have significantly reduced the original 933 ms average and 1061 ms p95 SteelEagle system latency shown in Figure 1. Figure 10 shows that connecting the Parrot Anafi drone to the CMU Cloudlet over T-Mobile IoT and using the PDrAW library for decoding the real-time video stream from the drone yields an average of 380 milliseconds of latency, with a p95 latency of 402 milliseconds (Table 1). That's a *more than two-fold improvement*. We also observe a significant reduction in variability when using PDrAW, with the standard deviation cut in half, resulting in more consistent performance. The new PDrAW configuration has shown clear benefits in terms of increased drone agility, particularly when tracking fast-moving objects. A drone using the SteelEagle system is now able to successfully track a person sprinting, which was not feasible at the time Bala et al published their paper.

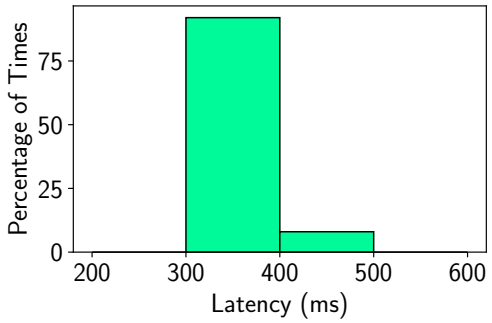


Figure 10: Latency Using PDrAW

Despite the significant reduction in latency, 380 ms is still substantial. The analysis in this report has taken an end-to-end view of the system latency and has not evaluated the piece-wise contribution of each component. How much of the 380 ms latency is because of the video decoding algorithm? What about the on-drone processing? Answering these questions is a prerequisite to further optimization of the system latency. What would it take to get down to 50 ms of end-to-end latency? We hope that this report has established the need for further research in analyzing the drone video stream latency that yields more latency improvements. These improvements would unlock even more compelling drone computer vision use cases.

References

[1] Bala, M., Eiszler, T., Chen, X., Harkes, J., Blakley, J., Pillai, P., and Satyanarayanan, M. Democratizing drone autonomy via edge computing. In *2023 IEEE/ACM Symposium on Edge Computing (SEC)* (2023), pp. 40–52.

[2] Community Project. FFmpeg. https://ffmpeg.org/.

[3] Community Project. OpenCV. https://opencv.org/.

[4] Parrot. ANAFI: Compact and resistant drone with a 4K HDR camera. https://www.parrot.com/us/drones/anafi.

[5] Parrot. Parrot ANAFI Product Sheet. https://www.parrot.com/assets/s3fs-public/2021-02/anafi-product-sheet-white-paper-en.pdf.

[6] Parrot. PDrAW. https://developer.parrot.com/docs/pdraw/overview.html.