4K High Definition Video and Audio Streaming Across High-rate Delay Tolerant Space Networks

Kyle J. Vernyi [*]
NASA Glenn Research Center, Cleveland, Ohio, 44135

Dr. Daniel E. Raible[†]
NASA Glenn Research Center, Cleveland, Ohio 44135

Audio and video streaming across delay tolerant networks are relatively new phenomena. During the Apollo 11 mission, video and audio were streamed directly back to Earth using fully analog radios. This streaming capability atrophied over time due to the gradual conversion to digital electronics accompanied with higher resolutions causing the required bit rates to outpace communication link performance. Additionally, 21st century space systems face the new requirement of interconnectedness. Delay Tolerant Networking (DTN) attempts to solve this requirement by uniting traditional point to point links into a robust and dynamic network. However, In order to avoid system bottlenecks, the High-Rate Delay Tolerant Networking (HDTN) implementation focuses on performance-optimization of the standards. This work extends the functionality of HDTN by implementing audio and video streaming, with the goal of demonstrating the practical application of high definition media streaming across space networks. A series of network topologies were created including simple point to point links and multi-node multi-hop networks. Test media in the form of prerecorded and live footage was streamed across the network. A set of objective quality metrics were established in order to measure the stream quality. A lunar network was emulated using a mixture of embedded ARM platforms.

[*]Software Engineer, NASA Glenn Research Center, 21000 Brookpark Rd
[†]Electronics Engineer, NASA Glenn Research Center, 21000 Brookpark Rd

I. Introduction

Media streaming is a complex science that is constantly evolving. The success of streaming in the terrestrial Internet has yet to be realized in space-based communications where networking itself remains a development effort. Classical streaming on the Internet uses assumptions of high connectivity, high bandwidth and low latency. These assumptions can not be made in space due to mobility, limited computational capability, and vast distances between assets. Recent research efforts towards a solar-system internet (see Figure 1) has given rise to a sufficiently optimized and mature networking approach known as High-rate Delay Tolerant Networking (HDTN) which can support streaming in this more challenging and generalized environment. In this work, a video streaming capability for HDTN is created, tested, and analyzed. Additionally, a set of objective performance metrics are presented such that future work (including other DTN implementations) can compare results.

HDTN is a C++ codebase built on top the Delay Tolerant Networking (DTN) standards which are in part codified by the Consultative Committee for Space Data Systems (CCSDS). CCSDS has also published a number of documents over the past decade aiming to standardize the transmission of media in space. This work utilizes most of the recommendations found in the CCSDS documents. New technology such as HDTN, optical communications, and encoding has enabled higher video resolutions than CCSDS had planned for. In these cases, HDTN will be tested to its maximum capability rather than the previous recommended cases. The CCSDS recommendations are taken into account in the test media preparation and the actual media streaming applications.

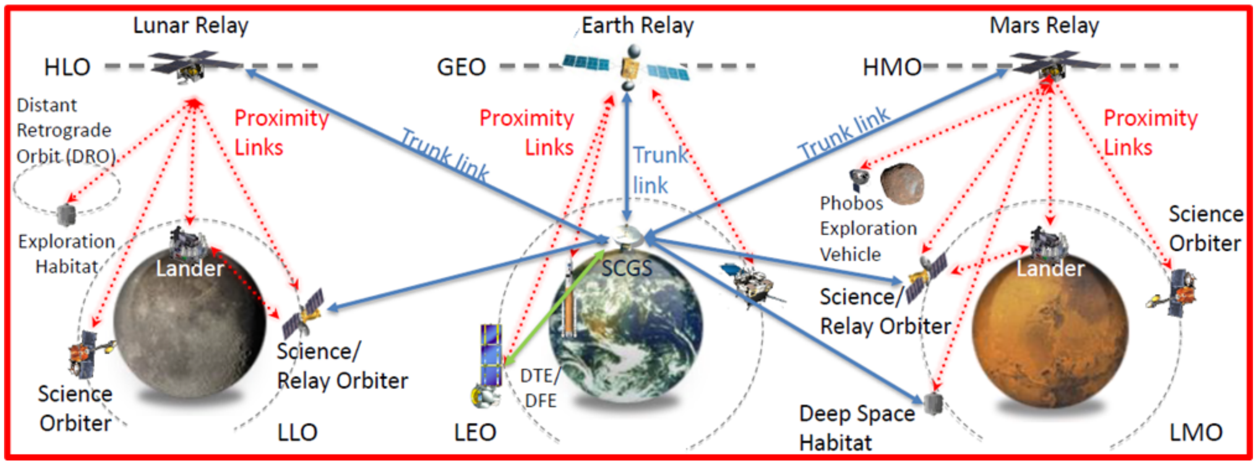

Fig. 1 Example topology of a solar system internet

A. LunaNet Emulation

One potential use case for HDTN is LunaNet. LunaNet is the proposed network architecture for supporting the Artemis exploration program. It seeks to provide a scalable interoperable network with global participation[1]. Due to the high performance requirements from the Artemis program, it is likely that HDTN will be one of the DTN implementations in LunaNet for the more demanding links. This work aims to implement one possible LunaNet topology involving a lunar rover. In this topology, the lunar rover sends a video stream of the lunar surface to observers on Earth. The network topology is shown in Figures 2 and 3. Depending on orbital mechanics, the rover may have a line of sight to a single or multiple nodes for a short period of time. Being able to handle these disconnections is the essence of a robust DTN. Using a program called Satellite Orbit Analysis Program (SOAP) the orbital dynamics of the system were simulated. Using these orbital dynamics, a contact plan is generated. This contact plan contains all the valid links in the network for a given time period. The contact plan is distributed to the nodes so that they can be aware of when there are link opportunities and planned disconnections.

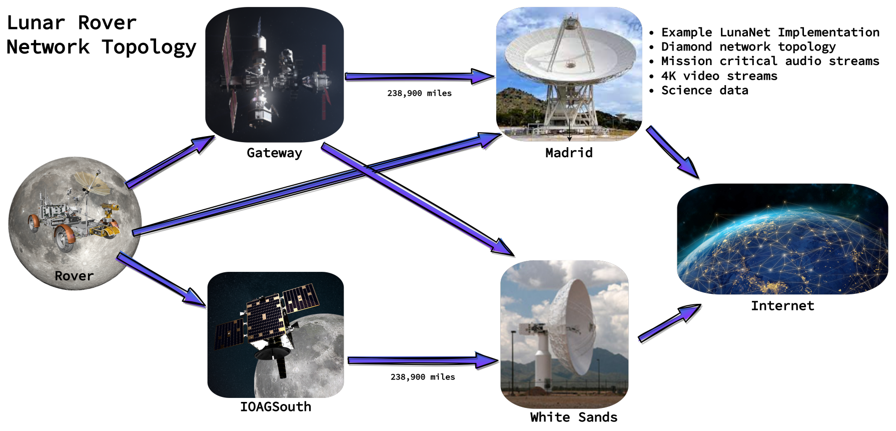

Fig. 2 Example LunaNet topology

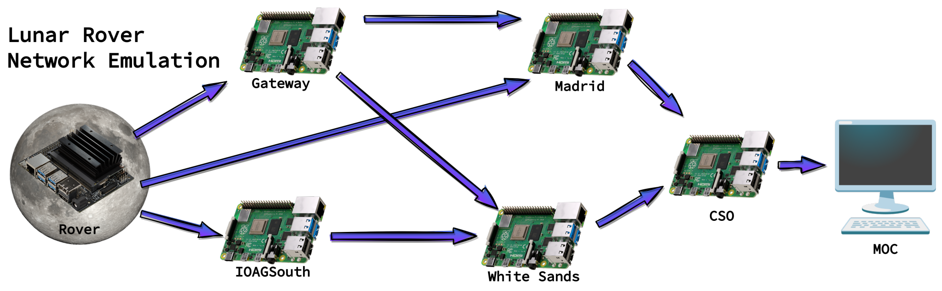

Fig. 3 Emulation of example LunaNet topology

II. Approach

There are multiple factors that play into the final quality of a media stream. These factors include video formats, encapsulation and transmission protocols, and real time requirements. The scope of this work was limited to the transport of a pre-generated media stream. If encoding and decoding were included, this work would be elevated to a system benchmark rather than a network test. Given the complexity of developing and using video encoders and their corresponding transport protocols, it can be assumed that a NASA mission is likely to use some form of third party library to generate video data. Indeed, the Mars Perseverance rover uses FFmpeg as the primary video encoding engine [2].

Two multimedia frameworks were used in this work; FFmpeg and Gstreamer. FFmpeg is a free and open-source software project consisting of a suite of libraries and programs for handling video, audio, and other multimedia files and streams[3]. FFmpeg was primarily used for encoding and decoding video streams in order to create test media. It was selected as the encoder and decoder due to its ease of use and popularity. Gstreamer is an open-source software library which uses graphs to create media processing pipelines [4]. Gstreamer was used to construct "pipelines" for media streaming. A pipeline is a customizable way to transform a piece of media into different formats, encodings, and transport protocols. For example, a pipeline may take an MP4 file and convert it to a live stream on the network.

A. Network Encapsulation

The network encapsulation protocol is another important consideration when attempting to stream media via both Internet Protocol (IP) and Bundle Protocol (BP). Video decoders generally require a consistent stream of data in order to properly decode the byte stream[5]. This consistent stream of packets is not easy in IP or BP networks. There are several different encapsulation protocols that provide different advantages and disadvantages. The UDP protocol is simple to implement but lacks robustness in the face of packet loss. TCP is not an option in high latency networks[5]. Another protocol is the Real-time Transport Protocol (RTP). Initial testing by CCSDS and Deutches Zentrum für Lift-und Raumfarhrt (DLR) has shown that RTP is more forgiving of interruptions and long latencies [5]. For this reason, RTP was chosen as the encapsulation method.

B. Real-Time Transport Protocol

Real-Time Transport Protocol is a scalable protocol that provides end-to-end delivery services with real-time characteristics [6]. Effectively, RTP prepends sequence numbers and timestamps to packets of media data so the receiving size is better equipped to reorder packets and handle network jitter. RTP is very widespread in the telecommunications industry. Widespread use in industry is critical as these services are aimed towards deployment into near space networks such as LunaNet[7]. Gstreamer provides complete RFC compliant RTP functionality and thus was chosen as the encapsulator and de-encapsulator. The CCSDS has experimented with MJPEG-2 as the transport protocol but RTP appears to be a more robust protocol for space links [8] and therefore this work focused entirely on RTP.

C. Streaming Applications

HDTN was designed to be a modular codebase. This provides users flexibility while designing their own applications. For this work two new C++ HDTN modules were written; `BpSendStream` and `BpRecvStream`. These modules were used in conjunction with HDTN's standard `hdtn-one-process` module. `BpSendStream` intakes an RTP stream from an outside source, bundles RTP packets, and sends the bundles to a BP network. `BpRecvStream` performs the inverse operation by intaking a stream of bundles, extracting the RTP packets, and forwarding the RTP stream to an IP network. Figures 4 and 5 show a high level view of `BpSendStream` and `BpRecvStream`. Notably, this work makes no attempt to generate the media stream using low level C or C++ code. The BP apps serve only to transport a pre-generated RTP stream.

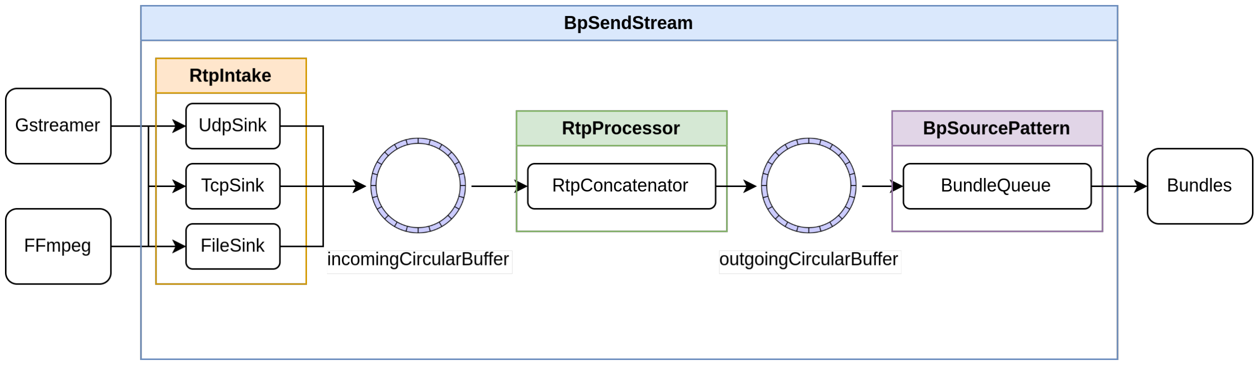

Fig. 4 Flow of data through BpSendStream

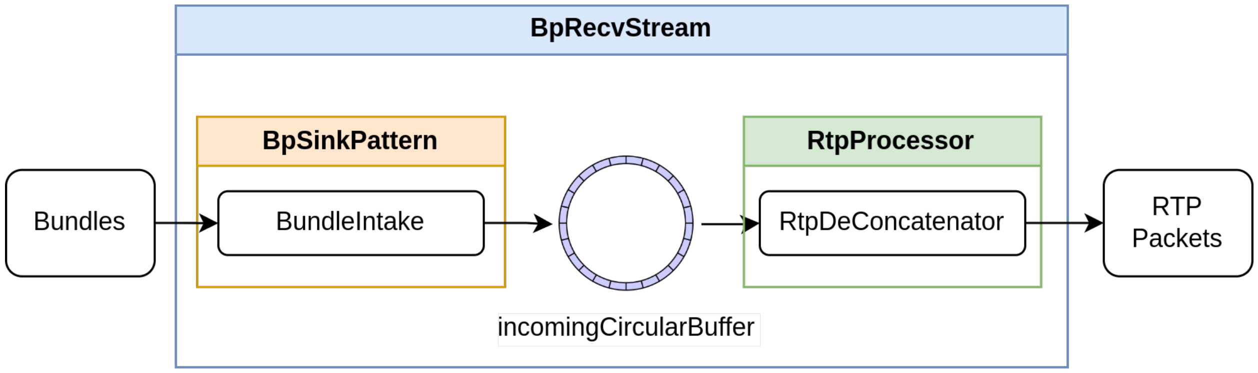

Fig. 5 Flow of data through BpRecvStream

III. Performance Metrics

Within the last three decades, the telecommunication and entertainment industries have become increasingly interested in objective video metrics in order to better measure the performance of encoders, decoders, and transport layers. Objective quality metrics for video are difficult to implement due to the inherently subjective nature of video. Previous experiments have been performed with RTP in a delay tolerant network but only bit rate performance metrics were given[9]. This work provides three performance metrics for video quality; peak signal to noise ratio (PSNR), structural similarity index (SSIM), and Video Multimethod Assessment Fusion (VMAF). All three metrics are full reference methods. Full reference methods compute the quality difference by comparing the original video against the processed video. In the most simple example, each pixel of every frame of the original video is compared against each pixel of every frame in the transmitted video.
Objective audio metrics are generally more simple to implement but face similar problems as video metrics when it comes to providing an accurate quality assessment. PSNR was used to measure the audio quality during streams.

A. Peak Signal to Noise Ratio

Peak Signal to Noise Ratio (PSNR) has been the telecommunication industry standard measurement for video quality for decades. The value is computed between every frame and then averaged. The computation is simple to implement. The biggest drawback of PSNR is that the final value does not correlate well to perceived video quality. In other words, a video with low PSNR may look higher quality than a video with high PSNR and there is no way of determining when this is the case. Despite this, PSNR was included as a performance metric due to its widespread use and ease of implementation. A PSNR above 35dB is considered “good”. A PSNR below 20dB is considered “poor” and usually indicates something went awry in the transmission. PSNR can be calculated as

$$\mathrm{PSNR} = 20\log_{10}(\mathrm{MAX_I}) - 10\log_{10}(\mathrm{MSE}),$$

where $\mathrm{MAX_I}$ is the maximum value of the encoding and MSE is the Mean Squared Error of the transmitted stream. The MATLAB implementation of this calculation for a `.flac` audio file can be seen in Listing 1.

B. Structural Similarity Index

Structural Similarity Index Measure (SSIM) is a perception based model that considers image degradation as perceived change in strucural information[10]. SSIM is an attempt to better model the human consumption experience. SSIM computation is more complex than PSNR but there are many different implementations freely available. It has also been an industry standard calculation since its introduction in the early 2000s. SSIM is ranked on a scale from [-1,1]. A score of 1 indicates a perfect similarity, a score of -1 indicates perfect dissimilarity, and a score of 0 indicates no similarity or dissimilarity. The SSIM score can be thought of as a first approximation to how a real person would rate the video feed, and teh one used here is the FFmpeg implementation.

C. VMAF

Video Multimethod Assessment Fusion (VMAF) is an open-source performance metric developed by Netflix. VMAF is a perception based metric that fuses together more simple metrics using a machine learning model. Netflix open sourced the tool in 2016 and it has been adopted rapidly across the entertainment industry. According the Netflix[11], VMAF linearly maps human opinion on a scale from 0 to 100. A score of 100 corresponds to a perfect viewing experience, a score of 70 would be a “fair” viewing experience, while a score of 20 or less is considered unacceptable. Importantly, a high VMAF score does not necessarily indicate a low bit error rate but rather a low impact of bit error rate such that the viewing quality is still excellent. The implementation of VMAF was provided by easyVMAF[12]. This Python library provides a wrapper around the VMAF C library.

IV. Testing

The HDTN video streaming plugins were tested against multiple types of network interference that a space based communication link may be subjected to. These include: varying bit error rate (BER), disconnections, reconnections, and delays. To emulate these parameters a Netropy 10G4 link emulator was used. The Netropy was used to enable robust testing of the streaming applications in a laboratory setting. The network topology remains the same for all of the following tests and can be seen in Figure 3. To model this network in the laboratory, two VLANs were constructed on either side of the Netropy as seen in Figure 6. All physical connections were wired Ethernet. Using this setup, assets on a particular VLAN can communicate perfectly and instantaneously with each other but assets on separate VLANs are forced to send data through the Netropy where the data can be modified. For the majority of testing, the Licklider Transport Protocol (LTP) was used inside of HDTN since LTP is the expected convergence layer for the majority of space networking traffic due to it's ability to handle long link delays.

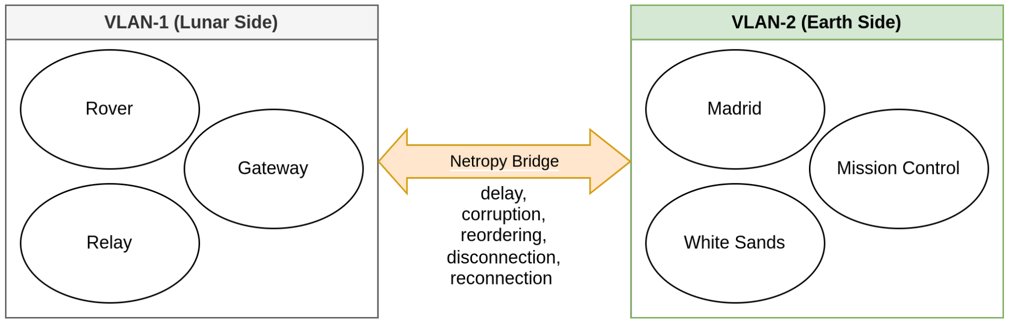

Fig. 6 Network emulation using Netropy and VLANs

A. Test Media Preparation

The test media conforms to the CCSDS recommendations for encoding. Recent advancements in the field of video encoding performance has made H264 encoding a more viable option for embedded platforms. The compression efficiency of H264 over traditional JPEG and MPEG is the main reason for this departure. As hardware H264 encoders and decoders become more prevalent, it is likely that future missions will implement H264 video for both computational and compression efficiency.

After an encoding scheme has been chosen, the CCSDS recommendations induce bitrate requirements. It has been shown that HDTN's performance meets the requirements even for high-resolution digital imagery on embedded ARM platforms. For this reason, this work utilized high resolution 4K imagery in order to provide a realistic stress test for the embedded implementations.

The `WaterBubble.mp4` video shows an astronaut playing with a 6 inch water bubble in zero gravity. The video was chosen due to its high level of detail which imposes a particularly high bitrate. This video represents the science data category in the CCSDS 766.2-B-2. Any artifacts from video transmission will be made obvious due to the intricate nature of the clip.

The `LuciaInterview.mp4` video shows an astronaut on the ISS speaking into a HAM radio to a group of school children on Earth. This video was chosen to test a synchronized audio and video stream. Any de-synchronization will be made obvious since the audio will not match the astronaut's face.This video represents the public affairs category in CCSDS 766.2-B-2.

The `AmmoniaLeak.flac` audio clip is a conversation between ISS astronauts and ground control. The astronauts are briefed on the status of an ammonia leak on the station. This audio clip represents the situational awareness category from CCSDS 766.2-B-2.

Notably, the pre-recorded test media is already encoded for the tests. The Jetson Nano is not performing real time encoding for any test except the live camera test. For these tests, Gstreamer only encapsulates the encoded video with the desired transmission protocol. A single frame from each video can be found in Figure 7.

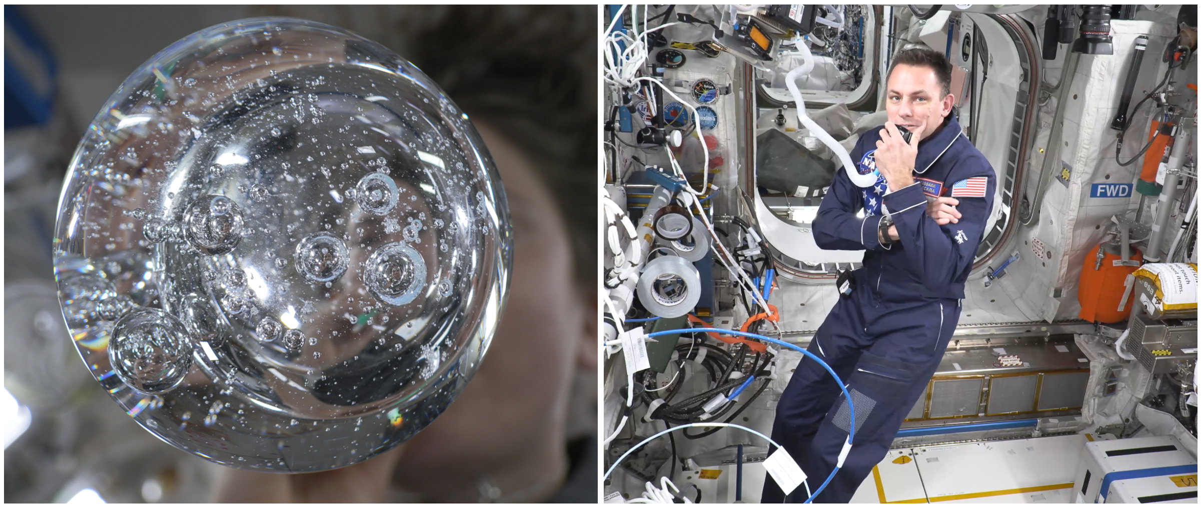

Fig. 7 Screenshots of `WaterBubble` and `LuciaInterview` test media

B. Bundle Size Testing

The first round of testing involves measuring the effects of varying bundle size on the quality of the stream. Bundles, like packets on the terrestrial Internet, have inherent overhead in their implementation. It is possible to reduce this overhead by reducing the number of bundles used to transmit information. Previous testing of video streaming introduced the concept of a "transparent gateway" in which RTP packets can be concatenated to any arbitrary size[5]. This allows for an adjustable bundle size which is highly desirable. Figure 8 illustrates how multiple RTP packets can be concatenated into a single bundle.

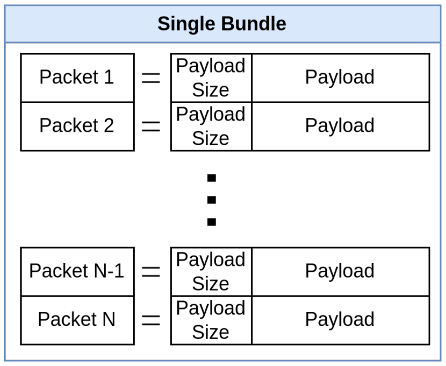

Fig. 8 Single bundle in the transparent gateway

For the first test, the number of RTP packets per bundle varied as seen in Table 1. The one way time of light delay was set to 1500 milliseconds inside the Netropy, providing a total round trip time of 3000 milliseconds. The `LuciaInterview.mp4` clip was tested first. Plots of the performance metrics for each test can be found in the Appendix A. The aggregate scores are presented in Figure 9.

A second test was conducted on the second video clip `WaterBubble.mp4`. Since the `WaterBubble.mp4` test clip has an appreciably higher bit rate, the bundles were allowed to be even larger than before as more RTP packets are generated per second. The bundles varied in size as seen in Table 2. Plots of the performance metric for each test can be found in the Appendix. The aggregate scores are presented in Figure 10.

RTP Packets Per Bundle	Max Bundle Payload Size (Bytes)
1	1400
5	7000
10	14000
20	28000

Table 1 Varying number RTP packets per bundle in the `LuciaInterview` test

RTP Packets Per Bundle	Max Bundle Payload Size (Bytes)
1	1400
20	28000
30	42000
45	56000

Table 2 Varying number RTP packets per bundle in the `WaterBubble` test

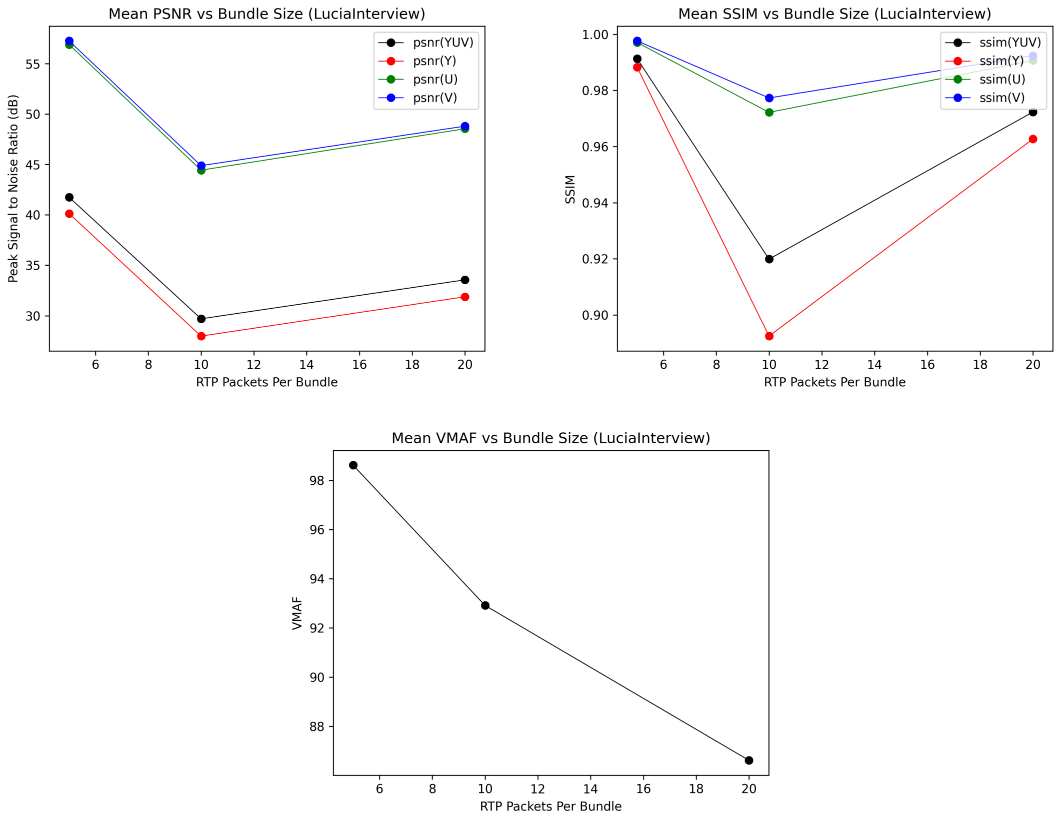

Fig. 9 Performance metrics plotted against increasing bundle size for `LuciaInterview.mp4`

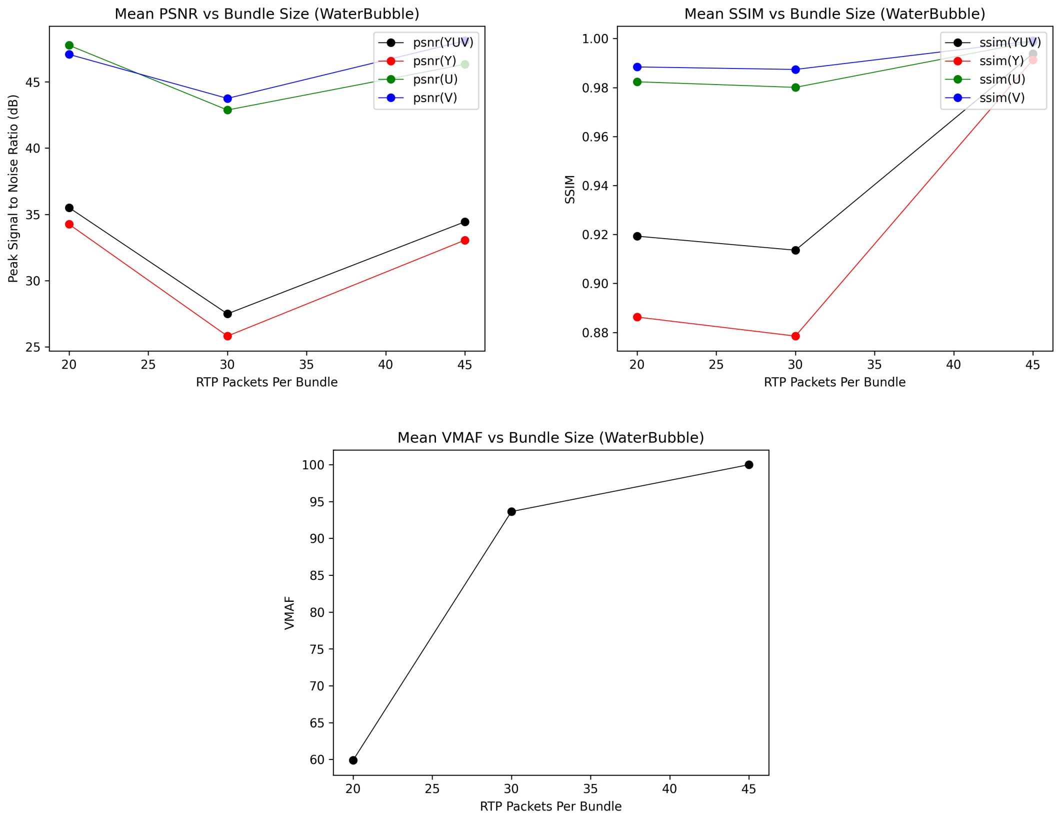

Fig. 10 Performance metrics plotted against increasing bundle size for `WaterBubble.mp4`

The results of the bundle size testing are interesting. The `WaterBubble` clip clearly performed better with increasing bundle size while the `LuciaInterview` clip performed better with decreasing bundle size. This could be explained in part by the higher bit rate of the `WaterBubble` test clip. Given that `WaterBubble` has twice the bit rate of `LuciaInterview`, it would benefit more from the increased bundle size as the overhead from BP and RTP are reduced.

The results for one RTP packet per bundle are not included since the resulting test clip was so corrupt that the performance metrics could not be calculated. This was expected as a bundle size of 1400 bytes with a bundle rate of thousands per second is demanding on the LTP convergence layer. The HDTN team is actively performing research on LTP optimizations so future work may benefit from staying within some ideal bundle size and rate.

C. Varying Bit Error Rate

Another round of testing varied the bit error rate of the data flowing from the lunar side of the network to the Earth side. This emulates a weak signal to noise ratio at the physical link layer. According to the Netropy user guide, "all packets that contain a bit error are discarded." This is an excellent way to test streaming using the LTP convergence layer as discarded bundles will be retransmitted. The `LuciaInterview.mp4` clip was used for this test. The plots for each test can be found in the Appendix in Figures 16 through 19 and the resulting cumulative performance can be seen in Figure 11. Notably, after the bit error rate was set to BER= 10^{-5}, there was a complete breakdown of Bundle Protocol, and HDTN failed to derive any data from the incoming bundles.

There is a clear decline in video quality as the bit error rate increases. The VMAF score provides an accurate objective measurement of what the video feeds. As the bit error rate increased, the amount of macroblocking and jitter increased correspondingly. Even as a bit error rate of 10^{-6} the video feed was intelligible. This indicates that the stream has some level of robustness likely provided by Bundle Protocol and RTP. Although the SSIM score appears to increase as the bit error rate increases, the increase is very small and likely within an error tolerance on the SSIM calculation.

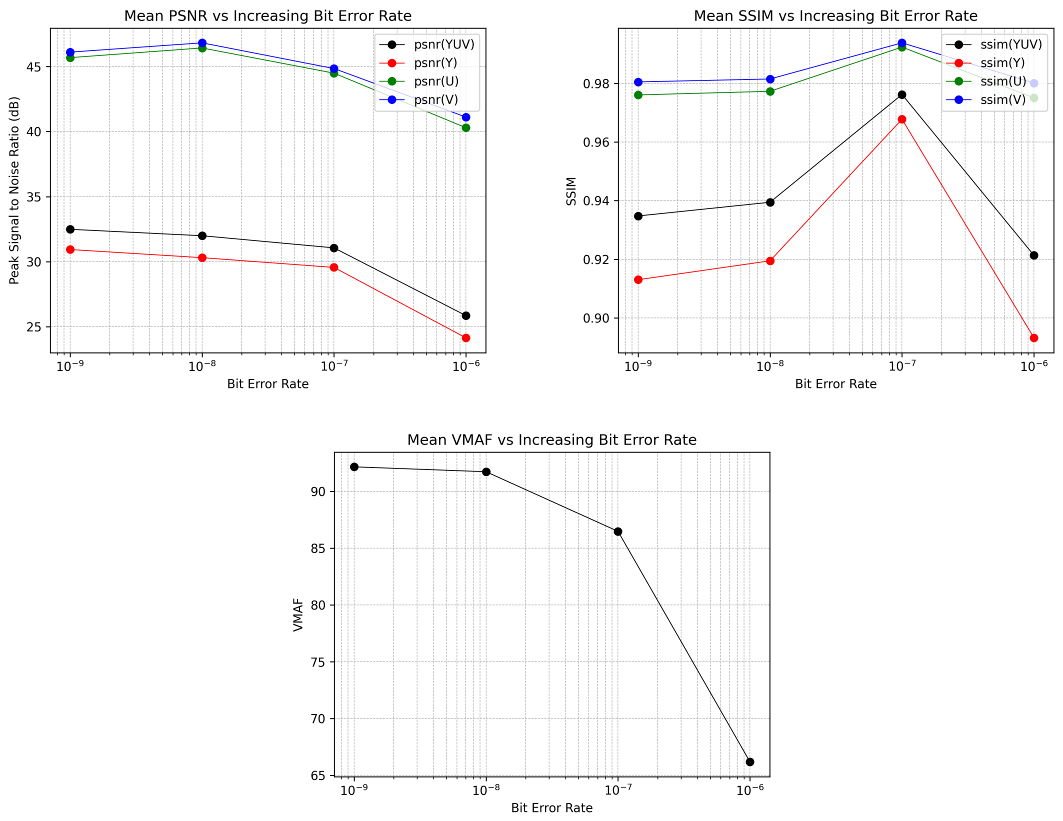

Fig. 11 `LuciaInterview` **performance metrics with 5 RTP packets per bundle, 1500ms delay, and varying BER**

D. Long Delay

This round of testing involved setting a very high one way time-of-light delay in the Netropy. Although the time of light to the moon is roughly 1300 milliseconds, the addition of digital electronics into the path may significantly extend this amount due to buffering and processing. Therefore by setting the one way delay to 10 seconds inside the Netropy, this test emulates LunaNet assets incurring substantial amounts of delay. The performance metrics can be seen in Figure 12. The long delay had no visible effect on the quality of the stream. The stream had two separate corrupted frames which are reflected in the metrics around frame numbers 900 and 1900.

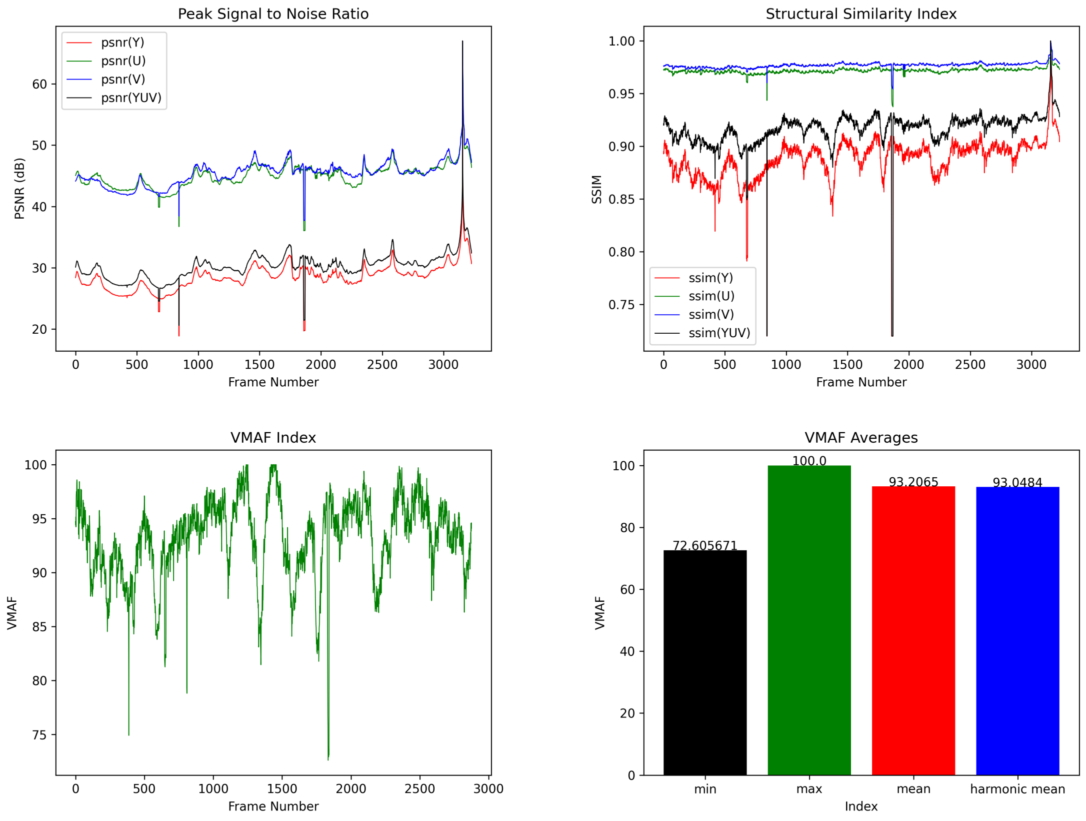

Fig. 12 `WaterBubble` performance metrics with 45 RTP packets per bundle and 10,000ms delay

E. Disconnections

Space networks are often subject to disconnections due to orbital mechanics. Therefore testing the performance of the HDTN streaming apps against disconnections is valuable. For these tests, the video feed was generated from a live camera source. A Raspberry Pi 4 recorded HD 1080p video at 30 frames per second using the Raspberry Pi HQ camera module. This is the maximum video quality the Raspberry Pi 4 GPU can provide. The video feed was streamed across the LunaNet network as before, with a 1500 millisecond delay. Physical layer disconnections were imposed between various nodes on the network. For this testing format, calculation of the performance metrics does not make sense since frames will be inherently missing and misaligned compared to the reference feed.

First, the "rover" node itself was disconnected by unplugging its Ethernet cable. Upon doing so, the video continued to play out for 1500ms due to the emulated delay in the network. Immediately after 1500ms, the video feed simply paused on the receiving side. HDTN reported `link down` events for the rover. Bundles continued to be generated and sent to storage, as expected. The Ethernet cable was reconnected after about 3 seconds after which the video feed was reestablished in two seconds. The internal GStreamer `rtpjitterbuffer` detected that there was a skew in the time delivery of packets and jumped ahead to catch up in time with the incoming RTP packets. In another test, the rover node was connected over WiFi to the Lunar VLAN. The WiFi router was unplugged during a stream. The same recovery time from before was seen.

This procedure was repeated and the amount of time spent disconnection varied from 3 seconds to over five minutes. The time to reestablish the video feed was the same regardless of the time spent disconnected. A fundamental issue with RTP applied to DTN was discovered here. Bundles that were sent to storage while a node was disconnected would fail to be converted to video due to RTP sequence number overflow. Since the sequence number field in RTP packets is only an `uint16_t` and over 1000 RTP packets can be generated per second in 4K video, the sequence number overflows every few minutes in high bit rate video feeds. Sequence number overflow is an expected behavior of the RTP protocol but presents an issue in the DTN environment where links can experience long disconnections. The sequence number overflow blocks recovery of the RTP packets since packets that are very out of order are dropped. An algorithm could be devised which also takes into account the RTP timestamp for the long term reordering of packets. However, this is a band-aid solution since timestamps can overflow as well. A quality solution to this problem would be another protocol on top of RTP that enables long term sequence tracking. This could be realized with a variable length header which has an ever expanding sequence number. Or, a DTN service could provide sequenced delivery and recovery at the bundle level. This issue with RTP caused packets that were sent to storage to be discarded upon their eventual reception to the final node. On the other hand, RTP sequence numbers and timestamps did perform very well in dealing with delay and small amounts of disruption. So the task of delivering a quality video stream was accomplished, but long term recovery of RTP packets sent to storage fails. This is non-ideal since much effort was spent in transporting those bundles and they are just discarded. But, for the video stream itself we may not be bothered by dropping old video if we have new video frames coming in.

F. Audio Streaming

The quality and reliability of audio transmission during mission critical operations is of utmost importance. The bit rate requirements for audio are much lower than video which makes transmission easier. The `BpSendStream` and `BpRecvStream` applications can be adapted to stream a variety of encoded media. In this case, AAC encoded audio was streamed across the network. The PSNR of the `AmmoniaLeak` audio clip after transmission through the LunaNet network was calculated to be 165.6381dB. In practice this translates to a perfect audio stream.

V. Conclusion

Going forward further testing will be conducted in the laboratory, as well as in the field with aircraft test flights and onboard the ISS with both RF and laser communications links. The results of these tests will be used to further refine the implementation of the HDTN media streaming plugins, which will be made freely available to the public to utilize. Furthermore the lessons learned from this research will be shared with the CCSDS committees to continue to improve the DTN related standards.

Appendix

A. Figures for `LuciaInterview`

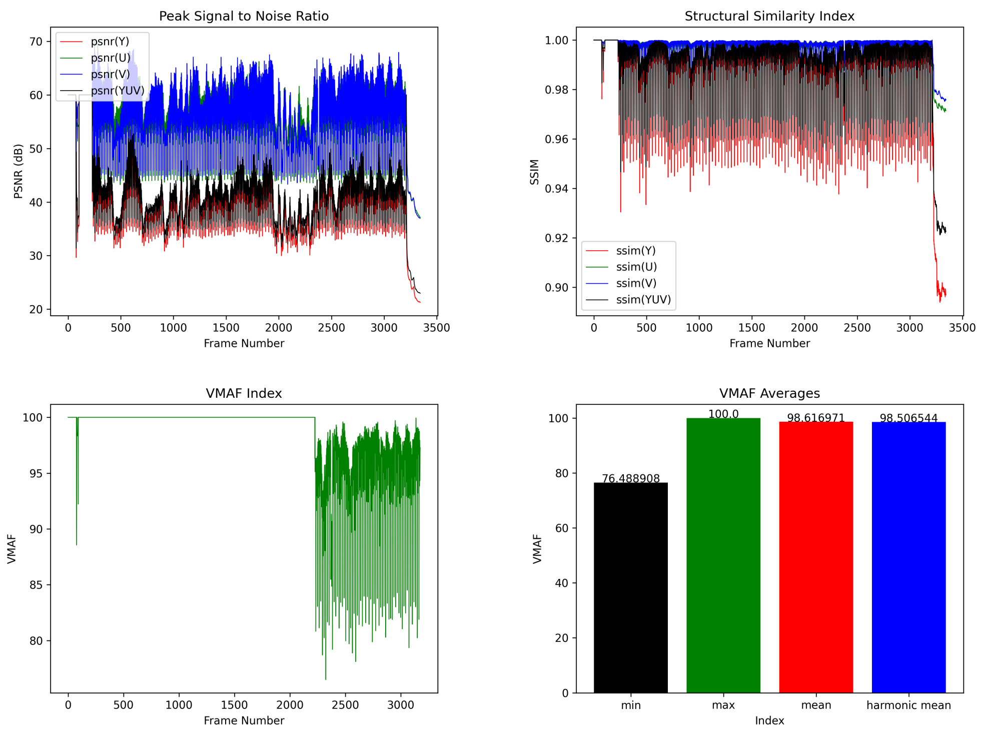

Fig. 13 `LuciaInterview` performance statistics with 5 RTP packets per bundle and 1500ms delay

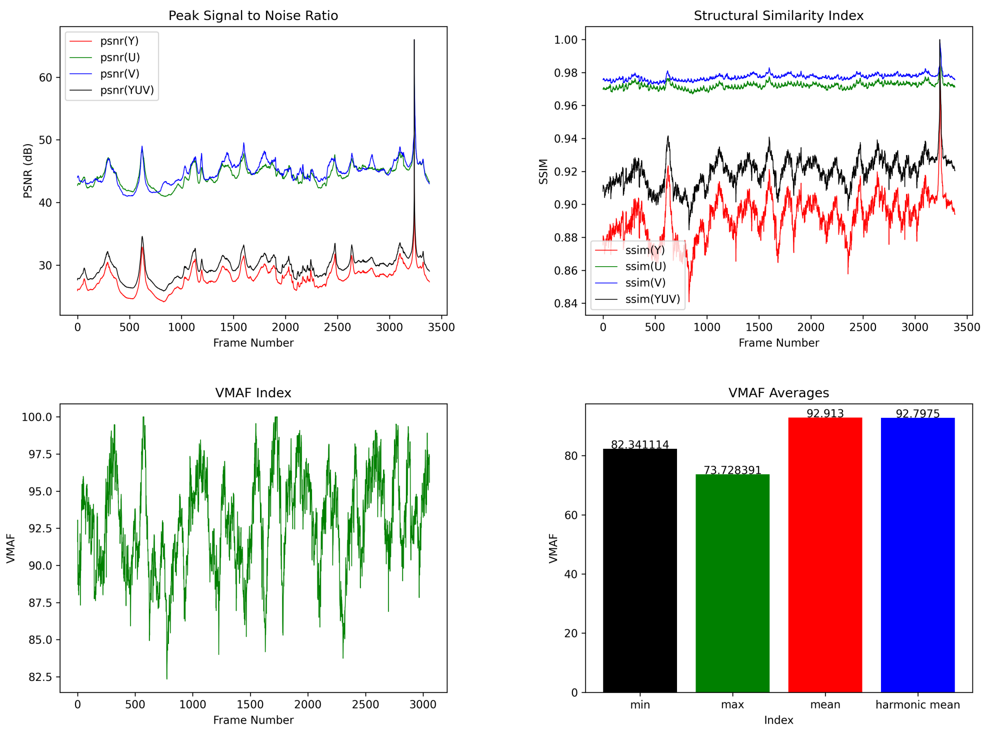

Fig. 14 **`LuciaInterview` performance statistics with 10 RTP packets per bundle and 1500ms delay**

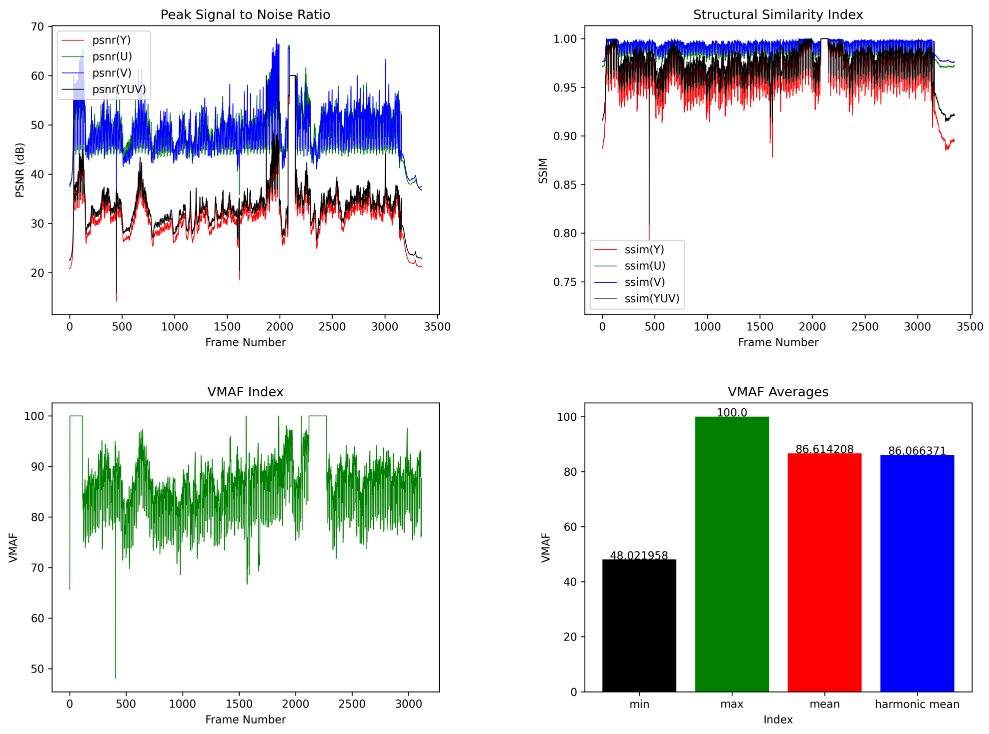

Fig. 15 **`LuciaInterview` performance statistics with 20 RTP packets per bundle and 1500ms delay**

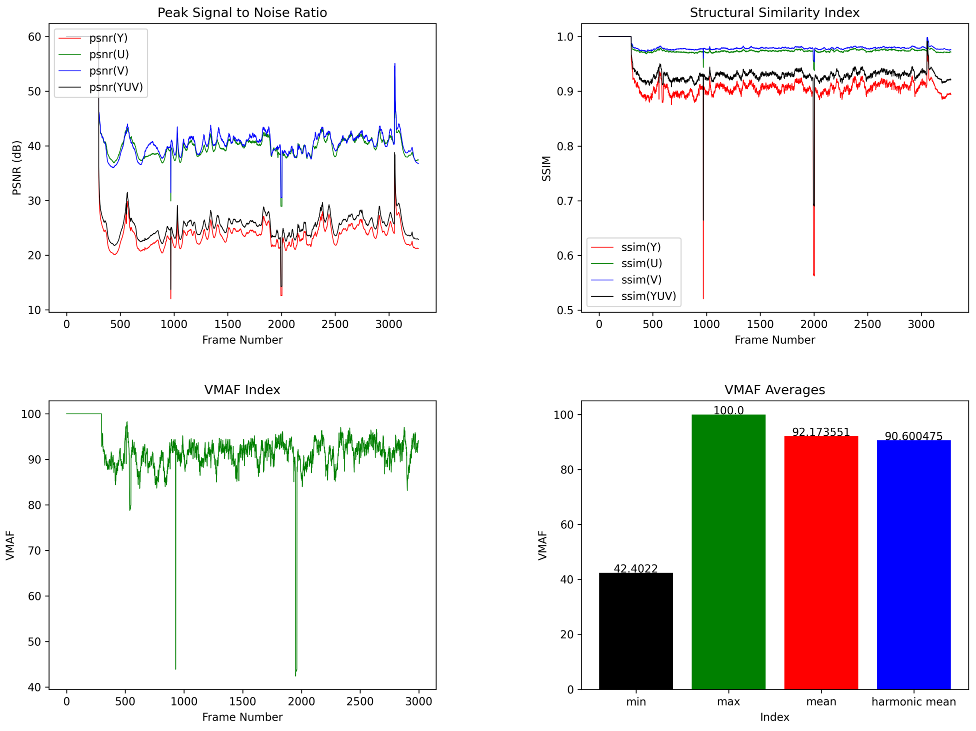

Fig. 16 `LuciaInterview` **performance statistics with 5 RTP packets per bundle, 1500ms delay, BER=** 10^{-9}

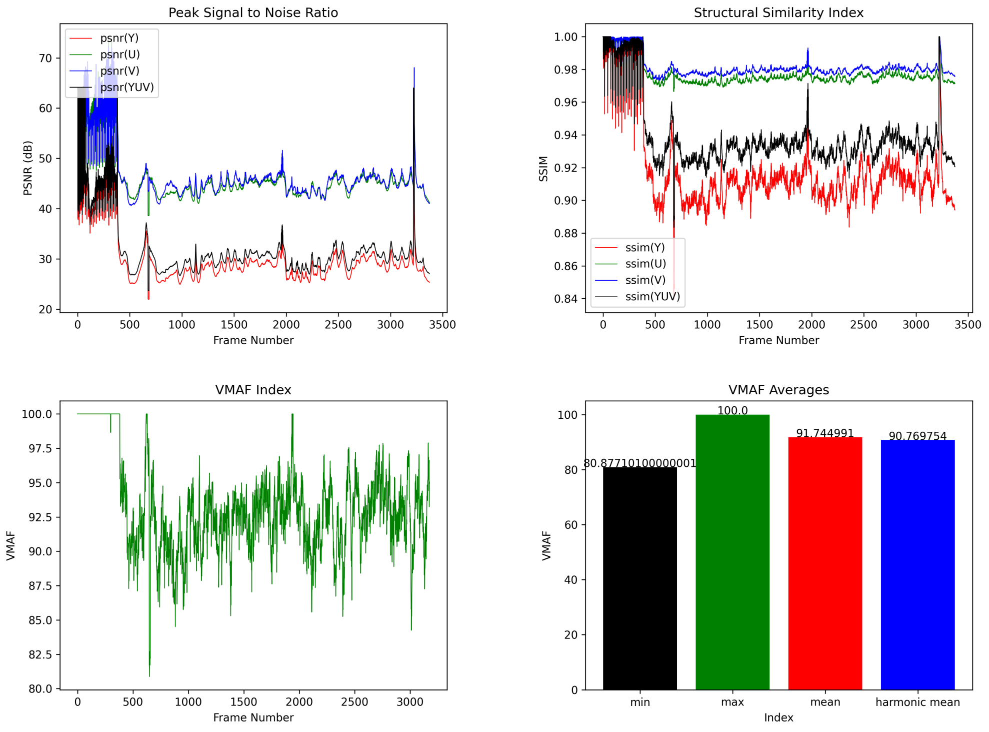

Fig. 17 `LuciaInterview` **performance statistics with 5 RTP packets per bundle, 1500ms delay, BER=** 10^{-8}

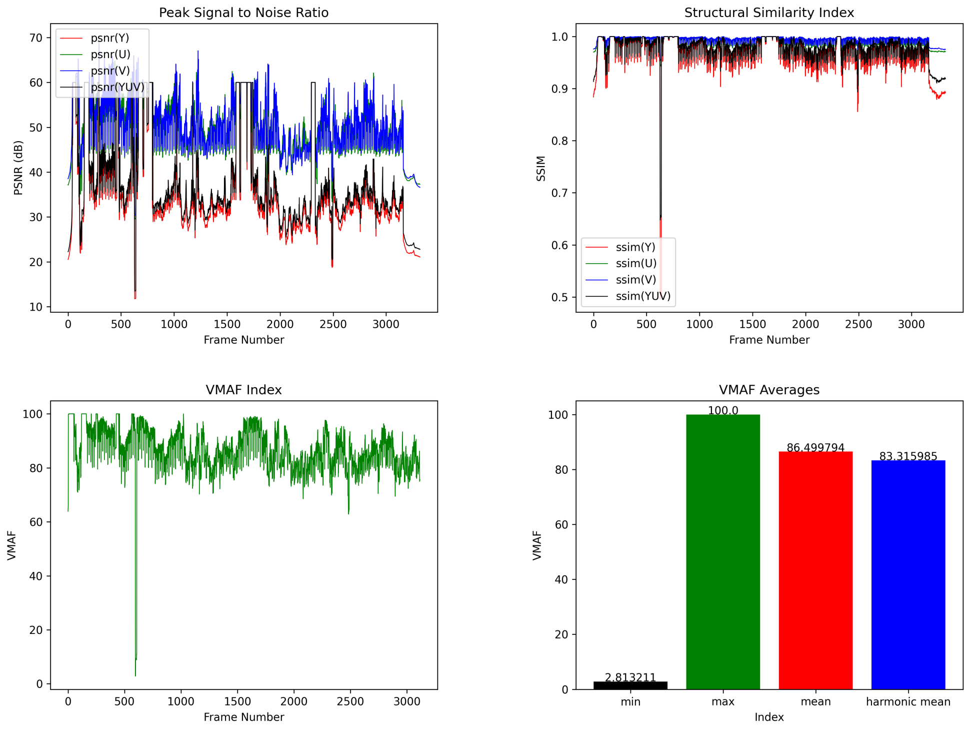

Fig. 18 `LuciaInterview` **performance statistics with 5 RTP packets per bundle, 1500ms delay, BER=** 10^{-7}

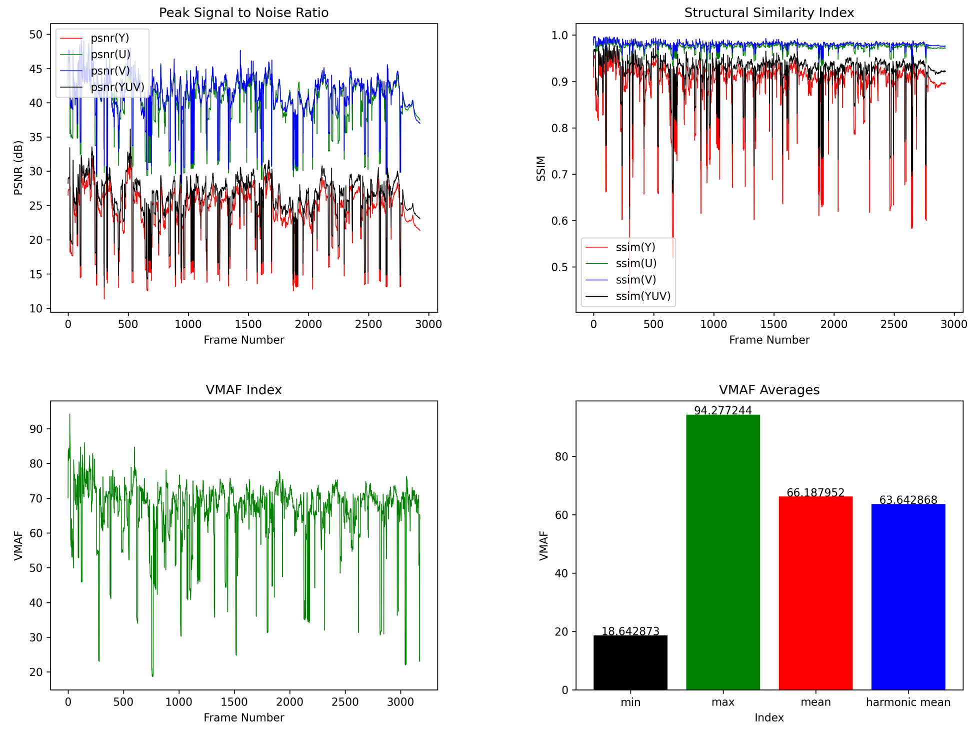

Fig. 19 `LuciaInterview` **performance statistics with 5 RTP packets per bundle, 1500ms delay, BER=** 10^{-6}

B. Figures for `WaterBubble`

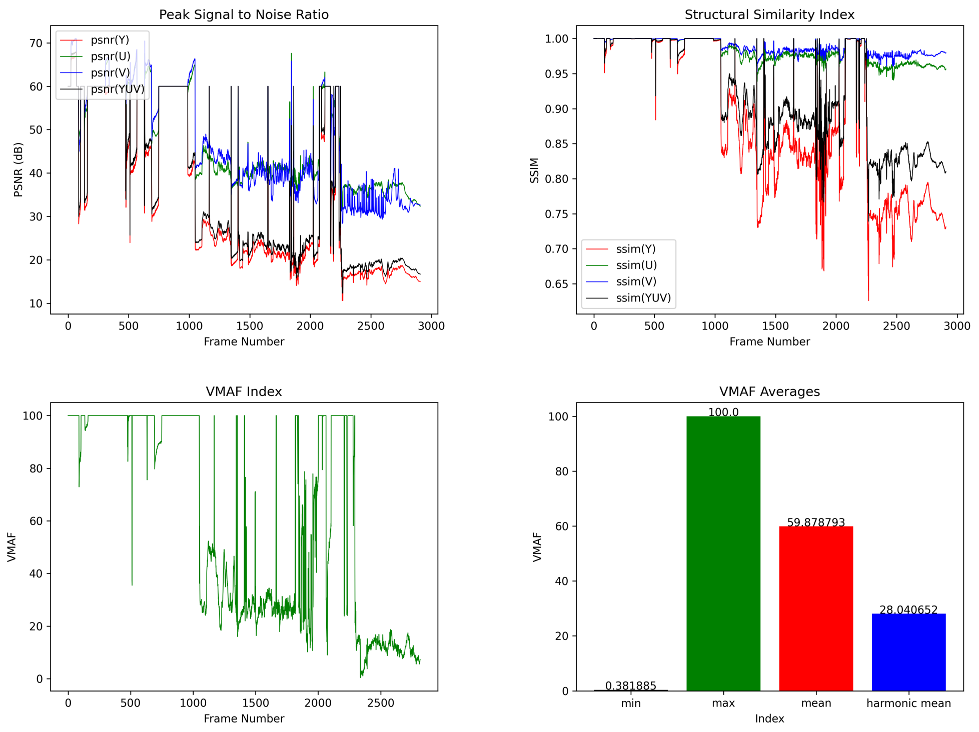

Fig. 20 `WaterBubble` performance statistics with 20 RTP packets per bundle and 1500ms delay

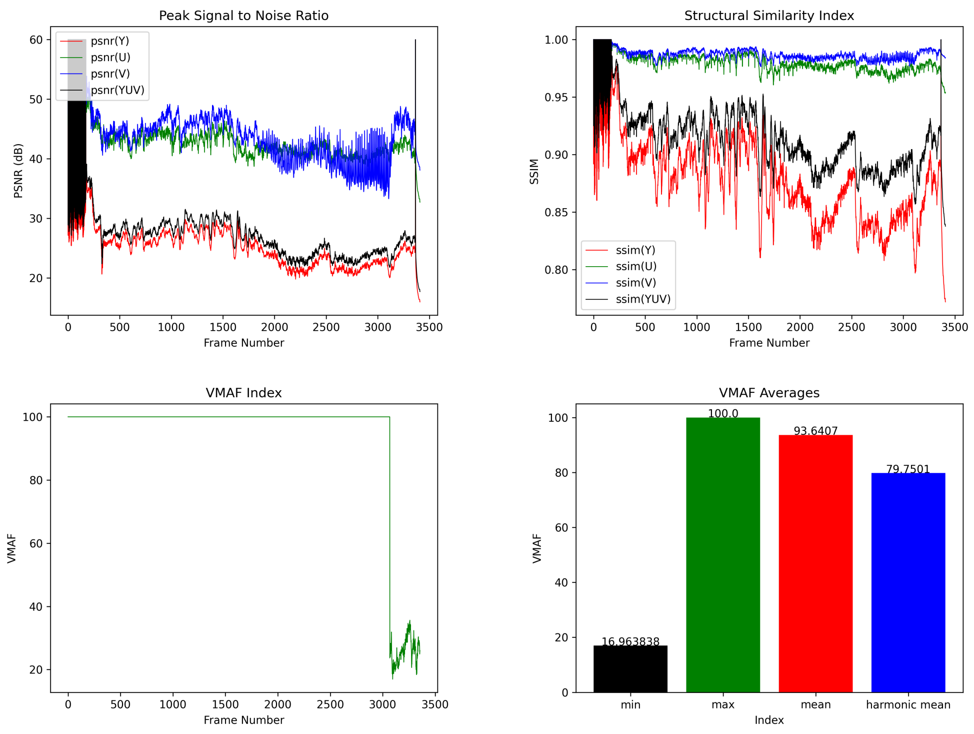

Fig. 21 `WaterBubble` **performance statistics with 30 RTP packets per bundle and 1500ms delay**

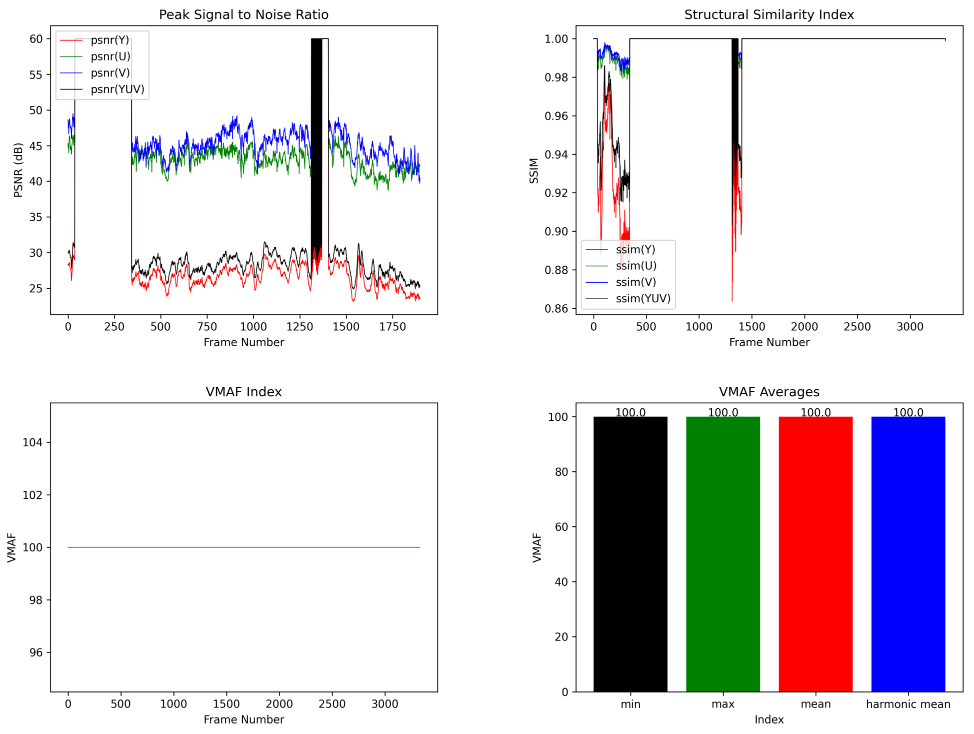

Fig. 22 `WaterBubble` **performance statistics with 45 RTP packets per bundle and 1500ms delay**

C. Code

```matlab
%% This file takes two .FLAC audio files as an input and calculates the
% peak signal to noise ratio (PSNR). The bitdepth of the flac file must be
% known so that MAXVALUE can be calculated as
% MAXVALUE = (2^bit depth) - 1
clc

MAXVAL = 2^24 - 1;  % 24 bit audio

transmitted_file="transmitted.flac";
reference_file="reference.flac";

[y_trans,Fs] = audioread(transmitted_file);
[y_ref,Fs] = audioread(reference_file);

% trim reference to start at beginning of transmitted
delta_length = length(y_ref) - length(y_trans);
y_ref = y_ref(delta_length + 1 : end);

%psnr calc
[R, C] = size(y_ref);
delta = y_ref-y_trans;
err   = sum((delta.^2))/(R*C);
MSE   = sqrt(err);
PSNR  = 20*log10(MAXVAL/MSE)
```

Listing 1   PSNR Calculation in MATLAB

Acknowledgments

The authors would like to thank the NASA Space Communications and Navigation (SCaN) program and the Delay Tolerant Networking (DTN) project for supporting this work. In particular the technical guidance provided by Josh Deaton (MSFC), Tom Basciano (JSC), Brian Tomko (GRC) and Alan Hylton (GSFC) is most appreciated.

References

[1] NASA, "LunaNet Concept of Operations and Architecture," Preliminary summary, National Aeronautics and Space Administration, 2020.

[2] NASA, "The Mars 2020 Perseverance Rover Mast Camera Zoom (Mastcam-Z) Multispectral, Stereoscopic Imaging Investigation," *Space Science Reviews*, Vol. 217, No. 1, 2021, p. 24.

[3] FFmpeg, `https://ffmpeg.org/`, 2023.

[4] GStreamer, "GStreamer," `https://github.com/GStreamer/gstreamer`, 2023.

[5] CCSDS, "SPECIFICATION FOR RTP AS TRANSPORT FOR AUDIO AND VIDEO OVER DTN," Draft recommended standard, The Consultive Committee for Space Data Systems, 2019.

[6] Schulzrinne, H., Casner, S. L., Frederick, R., and Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications," `https://www.rfc-editor.org/info/rfc3550`, 2003. https://doi.org/10.17487/RFC3550.

[7] Israel, D. J., Mauldin, K. D., Roberts, C. J., Mitchell, J. W., Pulkkinen, A. A., Cooper, L. V. D., Johnson, M. A., Christe, S. D., and Gramling, C. J., "LunaNet: a Flexible and Extensible Lunar Exploration Communications and Navigation Infrastructure," 2020, pp. 1–14. https://doi.org/10.1109/AERO47225.2020.9172509.

[8] "DIGITAL MOTION IMAGERY," Recommended standard, The Consultive Committee for Space Data Systems, 2021.

[9] CCSDS, "CONCEPTS AND RATIONALE FOR STREAMING SERVICES OVER BUNDLE PROTOCOL," Information report, The Consultive Committee for Space Data Systems, 2018.

[10] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E., "Image Quality Assessment: From Error Visibility to Structural Similarity," *Image Processing, IEEE Transactions on*, Vol. 13, 2004, pp. 600 – 612. https://doi.org/10.1109/TIP.2003.819861.

[11] Bampis, C., Novak, J., Aaron, A., Swanson, K., Moorthy, A., Cock, J. D., and Li, Z., "VMAF: The Journey Continues," https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12, 2018.

[12] gdavila, "easyVmaf," https://github.com/gdavila/easyVmaf, 2023.