A Hybrid Network/Tranport Protocol for Low-Speed Satellite Transceivers

Alex Wulff
Harvard John A. Paulson School of Engineering and Applied Sciences
Cambridge, MA, USA
awulff@college.harvard.edu

Hong-Long (Kit) Nguyen
Harvard John A. Paulson School of Engineering and Applied Sciences
Cambridge, MA, USA
honglongnguyen@college.harvard.edu

Abstract — **We present a lightweight approach to satellite communications that guarantees packet delivery and transmission of a stored buffer of data. Our efforts build upon the work of the open-source OpenLST satellite transceiver by adding a hybrid transport/network-layer protocol designed to run with extremely limited processing and memory constraints. Our approach centers around robust packet re-transmission schemes to guarantee successful packet delivery in a high-bit-error-rate environment. Acknowledgements containing information on lost packets flow back to the satellite from the groundstation, which are used in the satellite transceiver to inform further data transmission. We have demonstrated that our protocol is successful in sending data payloads to a groundstation during a typical pass in a simulated environment.**

I. Introduction

Ever since 2010 when NASA introduced the CubeSat Launch Initiative (CSLI) offering research-oriented spacecrafts a free/extra low-cost launch share to orbit through the Educational Launch of Nanosatellites (ELaNa) missions, getting to space has never been more accessible [1]. CubeSats are cuboid-shaped nanosatellites comprised of up to 12 multiples of 10 cm cubic units, or "U"s, each weighing up to 1.33 kg [1]. The Harvard Satellite Team branch of the Harvard Students for the Exploration and Development of Space (SEDS) – which we are part of – is currently developing a 3U CubeSat that we hope to launch into low Earth orbit (LEO) through the ELaNa program within a couple years' time.

Countless challenges are associated with building an amateur spacecraft, but the one we concern ourselves with in this paper is the challenge of transport-layer communication. For the harsh conditions of space where TCP falls short (reasons for which are described in Section II), many satellite transport layer protocols have been developed, most notably the S (SCPS-TP) and the open source Cubesat Space Protocol (CSP) [2].

Due to inevitable budget-induced hardware and software constraints (to be discussed in Section III), however, our communication system cannot use available transport layer solutions. Instead, using the low-cost OpenLST satellite transceiver available, we will build our own transport layer solution supporting packet chopping, reassembly, acknowledgments, and retransmissions tailored to the demanding parameters of our CubeSat link such as high bit error rate, variable round-trip times (RTTs), and limited pass time (time satellite flying over in line of sight for communication).

II. Problem to Solve

To better understand the problem at hand, we will divide our challenges into two categories and discuss how the otherwise universal TCP protocol fails to deliver for our circumstances.

A. Space Challenges

- High bit error rates (BERs): CubeSat links experience high BER (maximum of ~10^{-5}) and consequently high packet losses due to free-space loss and interferences from atmospheric and ionospheric effects. TCP's congestion control by reducing the window size (WS) in response to these packet losses would be completely counterproductive [2].
- Limited pass time: Due to their short flyover paths, LEO spacecrafts can only establish connection with their ground stations for 5-15 minutes at a time. Within that timeframe, the CubeSat may need to dump all its vital telemetry and data as well as receive new commands from the ground station [3]. Lengthy handshakes and frequent acknowledgements of TCP will wastefully cut into the precious transmission capacity.
- Variable RTTs: As LEO satellites moves through the horizon, their distance from the ground station will vary and this would not be compatible with TCP's fixed RTT assumptions.

B. Technology Limitations

As mentioned in Section I, general-purpose satellite transport protocols do exists, and some which are even available for free. However, these protocols require flight hardware resources that we do not have access to due to financial limitations.

Our prospective launch provider, NASA ELaNa, mandates and gives priority to CubeSats that use commercial off-the-shelf hardware with flight heritage (has been successfully deployed in space before). Due to the nature of limited supply and demand, such flight-proven hardware is expensive.

Thankfully, for satellite communication in particular, satellite imaging company Planet Labs open sourced their low-cost flight-proven transceiver design called OpenLST. Solutions other than OpenLST are financially prohibitive, so we are forced to work with all the pros and cons that come with this technology (the details of which are outlined in the next section). Most notably, the software provided with OpenLST does not offer a transport layer solution, which is where our work comes in.

III. Background, Motivations, and Prior Work

Despite the large quantity of man-made satellites currently in orbit around Earth, very few open-source hardware and software solutions for satellite data transceivers exist. Such open-source designs are crucial for university satellite clubs, as these clubs operate on a limited budget and oftentimes cannot purchase commercial off-the-shelf (COTS) transceiver hardware. The designs that do exist mainly offer a means of rudimentary packet transmission but fail to deliver an end-to-end solution to ensure the successful transmission of large quantities of data over a low-speed link.

An example of such an open-source low-speed transceiver is the OpenLST system by Planet Labs, as described before [4]. OpenLST provides the hardware and software necessary to send 255-byte packets from one radio to another. This system fails to deliver a transport-type protocol that divides data and ensures retransmission such as with TCP. Our work extends the OpenLST software to include these features. This system only has around 150 bytes of stack memory and 160 bytes of heap memory remaining for custom implementations, so our protocol was constrained to operate in this memory-limited environment. This inherently limited the number of features we could include in our protocol; for example, complex stated protocols such as those in use with TCP are infeasible. Similar incompatibility due to memory constraint arises if we try to implement general-purpose satellite transport protocols like SCPS-TP and CSP.

Nevertheless, we can take inspiration from how these space protocols combat satellite link challenges. In particular, we focus on techniques employed by SCPS-TP. To combat against high BERs, SCPS-TP employs an "end-to-end header compression that is robust against packet loss," so that "packets [coming] after a lost packet can still be received" [5]. In addition, the SCPS standard also provides an explicit corruption response to indicate that disruptions in the link are due to corruption instead of congestion [5]. But most crucially, to address both high BER and limited pass time, SCPS-TP has a Selective Negative Acknowledgment (SNACK) option [5]. By sending only occasional ACKs that explicitly lists out the specific packets lost, SNACK satisfactorily indicates retransmission needs while minimal impacting transmission speed. For its simplicity and efficiency in solving two problems at once, the idea of SNACK is also among one of the top features of our custom protocol design.

IV. Proposed Approach, Novelty, and Secret Weapon

We propose a hybrid transport/network-layer protocol to ensure reliable data transmission from the satellite to the groundstation. The payload size of the OpenLST hardware and software stack is limited to ~245 bytes, so our implementation needs to handle chopping and reassembly of data in addition to packet retransmission. The overall scheme of our protocol is as follows:

1. The groundstation initiates a connection by sending a dump_status packet to the satellite
    a. If the packet is lost or corrupted in transmission and the groundstation receives no response, it will automatically re-transmit the packet after a set interval.
2. The satellite receives the dump_status packet and begins by sending the first window of window_size number of packets. The satellite stores the furthest index sent of its data buffer.
3. The groundstation receives the window of packets. Each packet contains the sequence numbers of all the other packets in that window in addition to its own sequence number, so the groundstation knows which packets it expects to receive. If no packets are lost, the groundstation sends an acknowledgement to the satellite indicating so. If packets are lost, the groundstation sends an acknowledgement to the satellite with the sequence numbers of the lost packets.
    a. If the satellite doesn't receive the acknowledgement after a period of time, it will assume that either all the sent packets were lost or the acknowledgement was lost. It will then re-transmit the window after this period of time. The groundstation will ignore duplicated packets if the acknowledgement was lost and send another acknowledgement.
4. The satellite receives the acknowledgement from the groundstation, and it stores the sequence numbers of the dropped packets. The satellite retransmits the lost packets first and fills the rest of the window by sending packets of new data.
5. The groundstation receives this window. If any of the retransmitted packets are dropped, the groundstation will send their sequence numbers again in another ACK along with the sequence numbers of any new dropped data packets.
6. The satellite receives the groundstation's ACK and repeats the process in step 4 until its data buffer is sent. When the last packet is sent, the satellite also sends a DONE message.
7. If the groundstation still needs any packets, it will send an ACK back with the packets it needs. If it has received all packets and the DONE message, it will acknowledge the DONE message and the transaction will be complete.
8. If the satellite receives a request to retransmit packets after it has sent the DONE message, it will retransmit these packets and then retransmit the DONE message. It will continue to serve retransmission requests from the groundstation until it receives an acknowledgement of its done message, at which point the transaction will be complete.

a. If the satellite receives no acknowledgement of the DONE message after a certain period of time it will retransmit the DONE message.

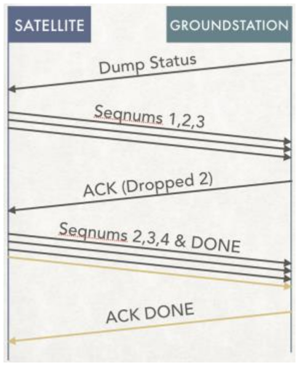

Fig. 1. Illustration of the proposed protocol

V. Intellectual Points

Despite the generality of the overall problem of sending data from one point to another reliably, the system used and goals of our satellite present unique challenges that require a novel data transmission scheme. We ultimately wish to transmit 100 kB or more from the satellite to the groundstation for any given pass of the satellite over the groundstation. Additionally, power constraints on the satellite and transceiver hardware dictate that our transmit power is limited to 1 W, resulting in a high bit error rate during a packet's journey from low Earth orbit (LEO). We also require that all data is transmitted successfully with no lost packets, which necessitates packet retransmission. As mentioned before, system memory constraints dictate that our solution must be relatively simple to implement in software.

Our communications protocol successfully operates within these provided constraints. In order to meet our goals, we had to make mathematically grounded-decisions about system parameters such as the window size, which in this system refers to the number of packets sent before waiting for an acknowledgement packet from the groundstation. Choosing the window size is solving a tradeoff problem between speed and robustness. As we increase window size, fewer ACKs will be sent and less time waiting for ACKs will be involved, but this comes at the expense of not knowing the connection status in the relatively long time in between the ACKs. Tracking the connection status, however, is fairly important for communicating with LEO satellites where tracking the fast-moving spacecraft over the horizon could be a challenge that requires frequent updates on whether the link is stable or connection needs to be re-established. To find the optimal speed-robustness tradeoff point, we included all our custom satellite link parameters to develop a precise mathematical model capturing the changes in total transmission time as we vary window size. The specifics and conclusions from our model can be found in the next section.

VI. Work Performed

We began our effort by defining the structure and behavior of our transport protocol, as described in Section IV. A key parameter still missing from our protocol is the window size, and for determining this we need to model the protocol's total transmission time as a function of window size, which is done as follows:

$$t_{\text{total}} = \left({}^{t}/_{\text{window}} + {}^{t}/_{\text{ACK}}\right) \cdot \#\text{windows}$$

$$t_{\text{total}} = \left(\frac{\text{data}_{\text{window}}+\text{data}_{\text{ACK}}}{\text{data\_rate}} + \text{RTT}\right) \cdot \left(\frac{\text{data}_{\text{total}}\cdot(1+2\cdot\text{drop\_rate}_{\text{packet}})}{\text{payload}_{\text{window}}}\right),$$

$$\text{where:} \quad \text{drop\_rate}_{\text{packet}} = \text{BER} \cdot {}^{\text{bits}}/_{\text{packet}} = 10^{-5} \cdot 2040 = 2.04\,\%$$

$$t_{\text{total}} = \left(\frac{255\text{B} \cdot \text{WS} + 255\text{B}}{375\ \text{B}/s} + 0.05\ s\right) \cdot \left(\frac{102{,}400\text{B} \cdot (1 + 2 \cdot 0.0204)}{240\text{B} \cdot \text{WS}}\right) \tag{1}$$

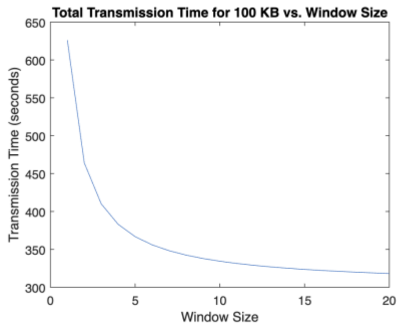

Fig. 2. Modeling the protocol's total transmission time as a function of WS

From the graph above, we see a clear optimal speed vs. robustness tradeoff point emerge at the elbow of the curve which occurs at window size of 5. We select this window size for the implementation phase for it offers relatively high speed at minimal costs of latency.

After clearly defining our data transmission scheme, we implemented it in software. The code for the satellite portion was written in C, as it needed to run inside the OpenLST software environment (which is also written in C). The groundstation transceiver is an identical OpenLST board, but we instead decided that it would be better to test our protocol over a wired link and simulate round trip time and packet loss parameters. This gave us more flexibility in testing our setup in all edge cases.

The OpenLST hardware and software support sending packets over a serial link instead of RF, so we decided to use that to send data to our simulated groundstation. The software for our simulated groundstation was written in MATLAB. This code initiates the transaction to the satellite by sending it a dump_status message and responds to all windows with ACKs. The groundstation randomly drops a user-configurable percentage of packets received to simulate packet loss. The OpenLST hardware has native cyclic redundancy check support to detect corrupted packets over the RF link, but since we're using a wired connection it was necessary to simulate the dropping of packets.

VII. Results and Discussion

Our simulated test environment successfully reached near the maximum theoretical speeds. Our test setup was able to transfer 100 kB over the simulated RF link in a time of 500 seconds given a window size of 5, a payload size of 100 kB, a round trip time of 50 ms, and a bit error of 10-5. This is 85% of the theoretical maximum speed. The speed loss can be attributed processing delay in the groundstation and satellite, in addition to other factors. We recognize that this test does not involve many subtleties of satellite communications, such as a non-constant bit error rate throughout the pass. To mitigate these effects, we selected conservative estimates for bit error rate, transmission speed, and round-trip time.

The code produced for this project can be found on GitHub. The C for the satellite can be found at http://bit.ly/2OVjN4d, and the MATLAB for the simulated groundstation can be found at http://bit.ly/2DOZAXL.

VIII. Conclusion

With an extremely minimal set of computational resources, we successfully designed and implemented a communications protocol capable of addressing the needs of our satellite system. We required that the protocol be capable of one-way transmission of a stored buffer of data with no lost information. Given the small packet size and relatively low RF power, this entailed devising a scheme to disassemble and reassemble buffer data and creating an acknowledgement protocol and a retransmission protocol.

References

[1] "About CubeSat Launch Initiative", *NASA*, 2019. [Online]. Available: https://www.nasa.gov/content/about-cubesat-launch-initiative/. [Accessed: 10- Dec- 2019].

[2] F. Davoli, C. Kourogiorgas, M. Marchese, A. Panagopoulos, and F. Patrone, "Small satellites and CubeSats: Survey of structures, architectures, and protocols," *International Journal of Satellite Communications and Networking*, vol. 37, no. 4, pp. 343–359, Apr. 2018.

[3] Cakaj, S. (2009). Practical Horizon Plane and Communication Duration for Low Earth Orbiting (LEO) Satellite Ground Stations. *WSEAS Transactions on Communications*, [online] 08(04), pp.373-383. Available https://pdfs.semanticscholar.org/ddc9/12b00accdba951712f3f15a4388ddea11b23.pdf. [Accessed 10 Dec. 2019].

[4] "Planet Releases OpenLST, an Open Radio Solution", *Planet.com*, 2019. [Online]. Available: https://www.planet.com/pulse/planet-openlst-radio-solution-for-cubesats/. [Accessed: 10- Dec- 2019].

[5] Scott, K. "SCPS-TP: A Satellite Enhanced TCP," *JPL TRS 1992* , Sep. 2004.