Creating an IP Router for Space to Ground Communications

David Rolenc
RT Logic
12515 Academy Ridge View, Colorado Springs, CO; 719-884-6344
dave@rtlogic.com

ABSTRACT

For smallsat ground station owners who are dissatisfied with the traditional and costly way of building stove-piped ground station solutions, why not simply treat the satellite as a member of the IP network on the ground? This solution allows the software on both the ground and satellite to dynamically change to meet the end-user needs, unlike the static approach of ground stations of the past. Using the internet protocol (IP) to implement space to ground communications allows flexible and affordable solutions. Leveraging currently available network stacks in conjunction with existing space communications standards brings the internet protocol to space, allowing standard client/server communications directly with the satellite by extending the ground network to space. Updating the satellite's software is accomplished by securely copying the software using readily available open source tools, such as secure copy (SCP.) Interfacing with the satellite's operating system is done using secure shell (SSH.) Software development for the ground station and satellite under this paradigm is just like traditional network software development, taking place at any time during the satellite's lifecycle. By developing a custom linux network driver, the network stack can be used to route packets to and from the satellite through a modem. The linux server on the ground has an IP address on its network that is on the same subnet as that of the satellite. Network Address Translation, which is built in to linux, can then be used to communicate with the satellite with traditional network programming techniques. Well accepted standards like CCSDS can be used to encapsulate the IP traffic that is transmitted to and received from the satellite. The satellite has built in software that performs the reverse operation of the ground.

INTRODUCTION

What makes a ground station expensive? The general answer is that ground stations don't take advantage of economies of scale. Most ground stations are built once per satellite or group of satellites in a very custom, stove-piped manner, and then they are only used for that mission. Most development inside of ground stations looks to solve an immediate problem. How do I get my payload for this satellite? How do I access the telemetry content for this satellite? How do I use this satellite's peripherals? How do I command this satellite? Engineers don't necessarily think about the larger problem of reuse and efficiency. We collectively have already solved this issue on the ground in terrestrial-based computers and networks. Think of a PC sitting on one's desk. That PC may have a very different mix of peripherals than any other computer, but the operating system has a set of standards it can use to access the peripherals in a seamless manner. It treats all cameras like cameras and all printers like printers and all keyboards like keyboards and all mice like mice. As long as the peripheral plays by the rules, it can be attached to the system and used almost immediately. The problem with satellite communications and more specifically ground stations is that there are currently very few standards dealing with how to build one. This leads to inefficiency and lack of reuse. There is no chance that any peripheral or piece of software can be reused as-is without a well-accepted set of standards. With the advent of small satellites, this may change. The push for small satellites has resulted in a demand for low cost solutions. One way to reduce cost is to encourage standards and reuse. That brings us to the topic at hand, using existing and already well-accepted standards to streamline the communications from the ground to space. This ultimately will make it easier to quickly and cheaply deploy and communicate with small satellites (or any satellites for that matter.) Standardized platforms coupled with greater demand leads to more plentiful and less costly solutions.

Standards

Which standards should the community use? On the ground, the Internet Protocol, or IP is the dominant solution. The internet makes use of the TCP/IP protocol to serve web pages, so there is no lack of developers who are well-versed in client/server programming utilizing IP. Looking at the space side, the Consultative Committee for Space Data Systems, or CCSDS, has a

large set of standards for ground systems and their interfaces to space systems. It's usually better to use standards that already exist and are well-accepted instead of making new ones, so using various CCSDS standards for communication to the satellite makes sense. See table 1 for the applicable CCSDS standards.

Table 1: Applicable CCSDS Standards

CCSDS Standard	Title
702.1-B-1	IP Over CCSDS Space Links
131.0-B-2	TM Synchronization and Channel Coding
732.0-B-2	AOS Space Data Link Protocol
231.0-B-2	TC Synchronization and Channel Coding
232.0-B-2	TC Space Data Link Protocol
132.0-B-1	TM Space Data Link Protocol

This paper deals with how to bridge the gap between the dominant terrestrial solution and the dominant space solution. The goal is to use commodity software development techniques on both the ground and the satellite to reduce costs. More to the point, no developer should have to know anything about CCSDS to make applications for the satellite or the ground interface to the satellite. That should be part of the standardized platform. Ideally, a majority of developers should operate within the realm of client/server development.

Design

If the end goal is to bridge the gap between IP and a satellite that accepts a CCSDS protocol, it stands to reason that a router is needed that can accept IP input and communicate with the space segment via CCSDS protocols. Fortunately, CCSDS has a standard for embedding IP within CCSDS frames. After the satellite receives the data, it needs to pull out the IP packet content and push the data on to the network stack. Similarly, when IP data flows from the satellite to the ground via CCSDS packets, the router on the ground should pull out the IP data and push the data onto the network stack for further routing. This router on the ground is a critical part of the solution, since it allows client/server communication from many hosts on the ground to potentially many satellites in space. With this solution, every satellite in space has an IP address, and every host on the ground wishing to communicate with it simply has to have a route through the ground to space router.

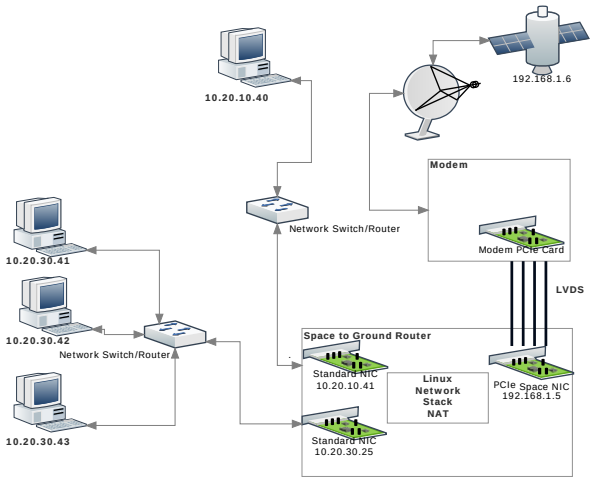

Figure 1: Ground Station Router

How to Make a Router

The linux operating system running on a commodity server can be used as a router. It has all the tools necessary to create complex routing structures. It can essentially bridge the gap between different network subnets. The most typical use-case of linux network routers is to route traffic between a private subnet and the internet making use of commodity network cards for data transport. This is known as NAT, or Network Address Translation.

The network card for communication to the space segment doesn't look like a typical network card, since it has to interface with a modem to communicate with the satellite using CCSDS protocols. The first implementation of this idea used a PCI express digital front-end processor card with LVDS inputs and outputs to interface with the modem. The linux driver developed for this card registered both as a linux character device and a network card. This was initially called a "Space NIC," and allowed the linux server to have an IP address for communication to the space segment. In this solution the ground segment is on a separate network subnet from the space segment. The hosts on the ground network need to be able to communicate with the space to ground router via a regular NIC on the same subnet, then the linux routing function routes the traffic to the space segment via the Space Nic.

Linux systems are split into user space and kernel space. Linux drivers operate within kernel space, while most other programs operate within user space. This address space separation provides some security to the operating system. User space programs are much easier to develop, so registering as a character device allowed

the transfer of data from the device driver to user space programs. The user space programs could then perform the required CCSDS translations and feed the result back into the device driver for further routing or transmission to the satellite. The linux device driver for the Space NIC does not have to be associated with a physical card; it can instead be a piece of software that performs the necessary CCSDS translation and forwards the result to an IP-based modem. Being a device driver that registers as a network card allows it to play in the linux network stack's routing tables.

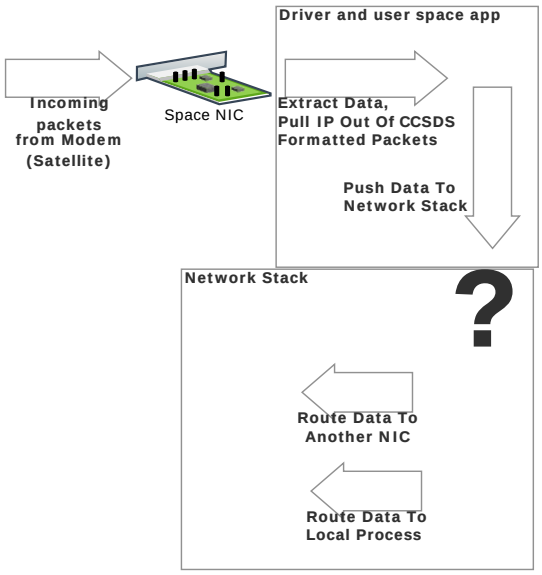

Figure 2: Example Incoming Packet Flow from the Satellite

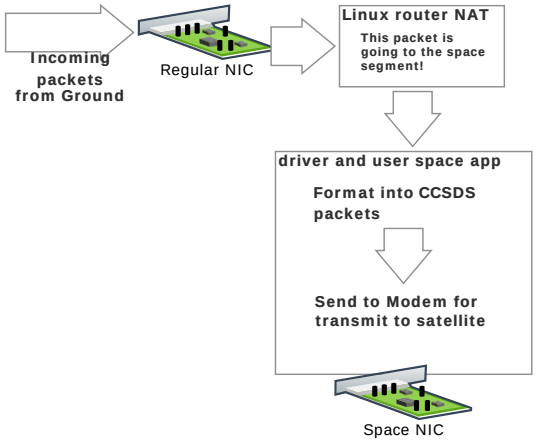

Figure 3: Example Outgoing Packet Flow from the Ground

Encryption Caveats

Packets on the space link are formatted according to CCSDS Internet Protocol Extensions (IPE) specifications. It may be desirable to encrypt this link, so some level of custom formatting for encryption may be needed. For example, AES encryption may require an initialization vector and key index to be passed in the clear in order to allow for decryption of the packet. Note that the key index is not the key itself, and the initialization vector does not need to be protected. Another option is to not encrypt the link at this level and count on other encryption mechanisms at different layers, like TLS/SSL. That is the way the internet works. There are many examples of how to set up secure client/server connections that follow best practices with TLS/SSL.

The Satellite Side

If the satellite is running linux, it can use a lot of the same code as the ground side router. The idea is the same, in that CCSDS packets are received from the ground with embedded IP packets. The IP packets are extracted and either routed through other communications channels or delivered to local applications. Local applications on the satellite can hold simple client/server conversations with the ground, since the device driver in the satellite abstracts away the complexities of CCSDS from the developer. The applications on both the ground systems and the satellite are reduced to simple client/server network applications that speak the already well-accepted TCP/IP, UDP or pick any other protocol that rides on top of IP. An example use-case is a satellite serving web pages using a stock Apache web server. The ground pulls up the satellite's web page using a standard Firefox web browser. Files can be transferred to the satellite using the standard scp (secure copy) utility. Administrators can use ssh (secure shell) to gain full access to the satellite's operating system. All the complexity of CCSDS is completely hidden, and the readily available client/server solutions can be used to provide powerful solutions both on the satellite and the ground.

Considerations for TCP/IP

TCP/IP is the protocol that the vast majority of the internet uses. TCP/IP provides guaranteed delivery (within reason) and guaranteed order. Internet traffic uses the HTTP protocol, but it rides over TCP/IP. TCP/IP is bi-directional and session-based, so a client and server establish a channel for communication. Each side must provide a "window," which is simply a buffer for data. When side A sends data to side B, side B must acknowledge receipt of the data to side A before side A can forget that data. After all, if the data doesn't arrive

to side B, side B will be requesting a retransmission of the data. This action is known as "advancing the window," and it makes room for more data to be transmitted between the client and server. Since communication latency can be great between the satellite and ground, any lost packets can cause significant backups in the data windows on either side of the TCP/IP conversation.

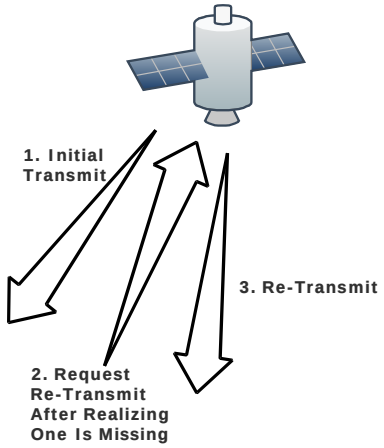

Figure 4: Missed Packet Retransmission

If throughput is valued, the TCP/IP data windows on both the satellite and ground should be configured large enough to ride through any network communications errors. The appropriate window size should be thought of in terms of data rate and round trip time, but is beyond the scope of this paper.

Forward Error Correction

One way to reduce the need for retransmissions of TCP/IP packets is to implement forward error correction on the link. The CCSDS TM Synchronization and Channel Coding standard provides numerous forward error correction techniques including Reed-Solomon Coding, Low-Density Parity Check Coding, Turbo Coding, and Convolutional Coding. The ability to correct the data instead of requesting a retransmission is crucial in getting good performance on the space to ground link.

TCP/IP Congestion Avoidance

There are many algorithms to regulate TCP/IP windows and other behavior. These are known as congestion avoidance algorithms. The main way that the network stack in any given system throttles TCP/IP date flow is by reducing or increasing the advertised TCP/IP window. When packet corruption occurs, the response by the congestion avoidance algorithms is to reduce the advertised window size. After all, if the link cannot keep up reliably at the current rate, what good is it to keep going that fast? Since TCP/IP is the backbone for the internet, many algorithms like this are mandated so that TCP/IP consumers and providers are good neighbors. One of the most problematic algorithms is the TCP slow start, which starts each TCP/IP session with a very small TCP/IP window, and gradually increases the window advertisement with each positively acknowledged packet. With large latency between the space and ground, it can take a while for the advertised window to be large enough to support a high throughput application. RFC-5681 mandates the use of TCP slow start[1], so it is not technically valid to remove it on space links.

The Nagle Algorithm and Delayed Acknowledgements

The Nagle algorithm is used to combine small packets into larger ones, provided there is unacknowledged data already in transit. The idea is to reduce overhead by combining the smaller packets into larger ones, but it can have negative effects on throughput and latency. It is recommended to disable the Nagle algorithm on applications in the satellite and on the ground unless the Nagle algorithm is shown to provide a needed benefit. Delayed Acknowledgements are similarly meant to reduce overhead by delaying TCP acknowledgements by up to 500 ms.[2] This behavior does reduce overhead, but is also reduces the rate of window advancement. When coupled with the Nagle algorithm, it can lead to undesirable behavior. The Nagle algorithm relies on acknowledgements to determine when to send the next data; if the acknowledgement is delayed, then the data that is to be transmitted is also be delayed.

UDP

User Datagram Protocol has a lot of attractive characteristics for space/ground communication. UDP does not guarantee delivery or order, but it is very efficient and fast. It is ideal for health and status data that periodically repeats, since if a packet gets missed, another will show up in a short time. It is not ideal for most payload data without a mechanism to request retransmissions of missing packets. In these cases developers find that they are recreating parts of TCP on top of UDP. There's nothing wrong with that approach, but it can lead to more custom code spread across the ground and space segments.

PGM

Pragmatic General Multicast has a negative acknowledgement mechanism. A receiver requests a retransmission only when it knows it is missing a packet. It does not guarantee delivery like TCP/IP, but it does maintain a data window for retransmission of

missed data. It is not required to hold on to the data until it is positively acknowledged by the receiver. PGM also maintains sequence numbers for the packets, so it can make sense of out of order receipt of data. This leads to the best of both worlds in that a receiver can request retransmission of a missed packet, but a slow receiver doesn't hold up the advancement of the data window. Since it is multicast, multiple receivers can consume data while not having a major impact on bandwidth from the satellite. With TCP/IP, a window is maintained for each client/server connection. In contrast, PGM can maintain one window per data source. This is very useful for payload distribution amongst multiple clients on the ground. If the payload happens to be important, the PGM windows can be very large to allow significant protection from missed packets.

Conclusion

Implementing routers on the ground that use existing and well-established standards to translate the common terrestrial network communications to standard space communications is an important step in reducing cost of ground station and satellite application development. Utilizing widely available developers who are well-versed in client/server development can lead to much more interesting work within the space community by focusing on the real problems instead of worrying about the communications. In addition, there is a great deal of client/server applications already available that can be used in satellite applications. Web servers and secure file transfer utilities are just the tip of the iceberg. Standardization of communication will lead to faster and cheaper deployments of satellites and their corresponding ground stations, and much more reuse in the industry.

References

1. Blanton, E. "RFC-5681 TCP Congestion Control," https://tools.ietf.org/html/rfc5681, September 2009.
2. Braden, R. "RFC-1122 Requirements for Internet Hosts - Communication Layers," https://tools.ietf.org/html/rfc1122, October 1989.