Satellite Data Acceleration

TCP-based Acceleration vs Streambed™

Warren Bailey, CTO
Acceleration Systems

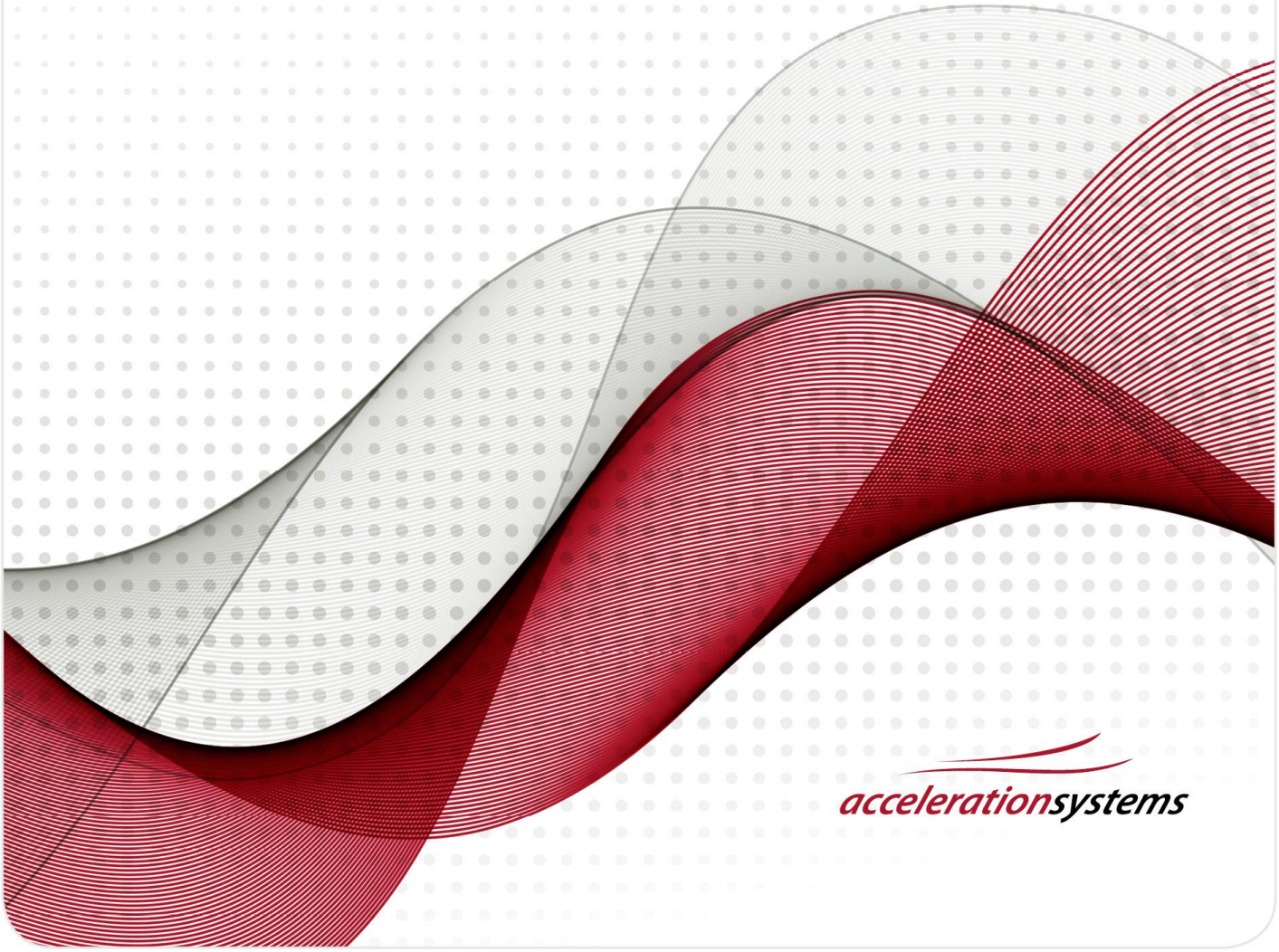

Table of Contents

Introduction

It was the Spring of 2006, and IP communications had finally made its way into the global VSAT hardware vendor product lines. It was all the rage. Users from around the world were flocking at the opportunity to deliver higher speed IP communications over a transport medium that until then had at best used Frame Relay or ATM. It was the dawn of a new era, a Golden Age of sorts, for an industry that until then was never considered a viable option for high speed communications. As operators began to deploy these newly introduced ground breaking technologies around the world, they started to notice some limitations with IP protocols that had not been encountered before. Though TCP had been introduced decades earlier and was working exceptionally well in the ever expanding Internet biosphere, it hadn't been deployed into a long latency application with unpredictable packet loss like Geosynchronous Satellite communications.

Later that same year, the SCPS-TP protocol was recommended and published by the CCSDS in an attempt to address some issues within the TCP protocol that were not working well with satellite communications. The premise of the SCPS-TP recommendation relied on a two way relationship between peers to measure data points like *bandwidth delay product* and *latency* across a link. Because SCPS-TP had to be so finely tuned for variable latency, the peers needed to have a firm understanding of the existing bandwidth delay product in order to adequately compensate for the link loss and delay when sending data over the TCP network.

After several years of pain and suffering through beta deployments and product revisions, VSAT hardware vendors began to recommend to some of their customers PEP (Performance Enhancing Proxy) solutions that attempted to address these inherent issues with TCP over Satellite. These third party hardware solutions often use exceptionally trivial techniques to decrease the impact of latency and total data sent "over the wire" in a satellite application:

1. SCPS-TP
2. TCP/UDP Header Compression
3. Lossy Image Compression
4. GZIP HTTP Compression

We'll examine each of these techniques in turn, and demonstrate that their benefit on a modern satellite network is questionable at best.

SCPS-TP

SCPS, or "Skips" as it is commonly referred to by the satellite industry, attempts to provide temporary relief from limitations in the TCP standard that are incurred when deploying it in long latency satellite networks. A peer to peer relationship is formed between two SCPS clients, and the link characteristics are monitored for packet loss and increase in latency. When the SCPS client sees a higher degree of latency, it changes parameters within the TCP protocol in an attempt to compensate for this loss. In certain situations, this can improve the characteristics of the link, but often times this mechanism is no better than what is now standard in commercial off the shelf software operating systems purchased today. With the development of the Internet and improving protocols, TCP stacks have become much more stable and ready to deal with longer latency related issues. Testing by NASA Ames Research center confirms that that SCPS in a PEP deployment had no tangible benefit past an ordinary TCP stack on a modern PC.[1]

[1] http://saratoga.sourceforge.net/draft-wood-tsvwg-saratoga/finch-pep-evaluation-ieee-aerospace-2013.pdf

TCP/UDP Header Compression

TCP and UDP packet headers contain information that is often repetitive. The idea behind this methodology is to detect and strip away this data, thereby transmitting less actual data over the transport medium. Though this methodology works, the bandwidth savings are often very minimal compared to the larger data payload being delivered. TCP and UDP overhead accounts for only ~10% of the transaction data, however there is a degree of latency introduced into the system when data is fed through the compression mechanism. This increased latency further impacts the performance of the network by affecting the total *bandwidth delay product* of the link. It should also be noted that no form of header compression will work with encrypted traffic, so if an encrypted IPsec tunnel is created over a network the resulting traffic cannot and will not be compressed. Compression algorithms remove redundancy, while encrypted traffic creates random patterns that must match at the opposite ends of the tunnel. Header compression is additionally useless for Layer 2 data which cannot be compressed in any way, shape, or form.

Lossy Image Compression

Web servers around the world host an array of images for download by clients, however those images are often not sized specifically to the application. The use of HTML and CSS allows web developers to resize the larger images in the browser, but the entire image still needs to be downloaded in order to display it. Lossy image compression techniques employ a downsampling algorithm, whereby the requesting client is sent an image file that has lower resolution than the original image received by the proxy. This lossy image technique is somewhat effective in saving bandwidth (~20% of total image size), though it should be noted this methodology is extremely available and can be deployed on commercial off the shelf hardware running a Linux-based Squid proxy with a trivial investment of time and energy.

GZIP HTTP Compression

As web technologies continue to evolve, some content providers have recognized the savings that can be generated by sending pre-compressed versions of files across the Internet to clients. One way to do this, is to implement GZIP HTTP compression and another is to utilize deflate. These methods can be helpful, though the web server receiving the request must be configured to respond to a GZIP or deflate request (it is not enabled by default on many popular web servers). As there are few websites on the Internet that want to take the time to configure this option, it is seldomly used. There is also the increase in time required to decompress the received data by the browser, adding to the latency of the overall connection and again affecting the user session. There have been several successful implementations of this methodology in the form of Internet browsers, most notably Google Chrome, so this functionality is available and very likely used in your networks today without adding any 3rd party product.

Why TCP is history and you shouldn't use it on your satellite network anymore…

In a TCP based session, the overall *delivery delay* for the link layer packet is a combination of *transmission delay*, *propagation delay*, and *processing delay*. *Transmission delay* is the frame size divided by the link speed. *Propagation delay* is the time it takes to traverse the entirety of the link. *Processing delay* is induced by both the sender and receiver. These factors have a cumulative effect on the quality of a TCP session over a long delay wireless link with unpredictable packet loss. When TCP sessions take place, receivers are not dealing with

singular packets but rather large blocks of data sent in chunks which need to be checked and reassembled for output into the client program.

TCP supports transparent segmentation, reassembly of user data, and handles flow and congestion control. TCP packets are acknowledged cumulatively as they arrive in sequence, out of sequence packets will cause duplicate acknowledgements to be created. The sending client detects a loss when multiple duplicate acknowledgements arrive (usually three in most TCP stacks) to show that packets were indeed lost and a retransmission is required.

As IP may reorder datagrams, the TCP protocol cannot instantly assume that gaps in the packet sequence indeed signify actual losses. When a TCP session becomes idle or acknowledgements are lost, TCP will detect these losses using *timeouts*. *Retransmission timers* are continuously updated using a weighted average of the previous round trip time measurements. And this is a very bad thing when you have variable latency and frame losses as a normal way of doing business in satellite.

As with any error detection, accuracy is critical as slowed or delayed timeouts slow down recovery and early timeouts can lead to redundant retransmissions. This characteristic of TCP is the reason your satellite network seems much slower than the bandwidth you allocate to your upstreams and downstreams during times of congestion.

The TCP protocol was never designed to recognize packet loss as packet loss, but rather assumes that ALL losses are congestion related. When losses are detected, TCP will not only retransmit the missing packet, it will also reduce its transmission rate assuming there is a router somewhere needing to drain out its queues. The TCP protocol also maintains a *congestion window*, an estimate of the number of packets that could be in transit without causing congestion in the network to occur. Additional new packets may only be sent if allowed by both this window and the receiver's *advertised window*.

The congestion window starts at packet one, with new acknowledgements incrementing it by one and doubling in size after each round trip. This is what is commonly known as the TCP *slow start phase,* and it is based on an exponential increase over round trip times. A *slow start threshold* is set to half of the value of the of the congestion window, the window is reset to one packet, and the lost packet is then retransmitted. The TCP slow start is then repeated until the threshold is reached after three round trips across the link.

Once the packet loss stops, the TCP mechanism is then free to begin opening the window size larger. This cumulative effect has dire consequences on TCP over satellite, as the session requires additional time to stabilize and begin operating normally.

By now you are probably thinking to yourself, if TCP is such a villain in this story who is the hero?

UDP is the Hero (almost)

UDP features far less overhead and the lack of acknowledgements enables it to utilize bandwidth MUCH faster and MUCH more efficiently. TCP is constantly looking for ways to handle congestion, while UDP is agnostic and sends as much as possible without regard to potential upstream issues.

But like every superhero, UDP does have a core weakness. That weakness is that UDP lacks error correction and retransmission capability. This weakness is acceptable for applications like

VoIP or live video, but unacceptable for any other purpose (files, email, web, etc.). This lack of error correction and packet retransmit has prevented deployment of UDP-based solutions. Fortunately for us (and the reader), there is an answer: Streambed.

The Real Hero is Streambed™

Streambed is a proprietary data protocol utilizing high performance UDP datagrams with a reliable error correction and retransmission scheme similar to TCP based protocols. By combining the best attributes of TCP and UDP, Streambed offers reliable world class data transfer speeds over transport mediums that are often very under utilized. When a Streambed session is established, an encrypted AES-256 bit tunnel is formed between the remote appliance and the secure Streambed cloud server network. This secure tunnel is entirely UDP based, meaning the limitations and challenges in TCP are completely non existent. The UDP stream is capable of consuming all available bandwidth resources allocated to the remote site, as the TCP slow start mechanism is not being used and data rates are not being manipulated. When packet loss is encountered, the reliable error correction mechanism within the Streambed protocol kicks in and resends data that was lost (just like legacy TCP).

The Streambed protocol eliminates the need for complex VPN deployments and other GRE type solutions, as the Point-to-Point relationship of the transmission tunnel allows for seamless integration into existing MPLS or VPN data networks.

Warren Bailey, CTO
Acceleration Systems
wbailey@accelerationsystems.com
(216) 468-5200 x217