Starlink Protocol Performance

Geoff Huston
APNiC
Low Earth Orbit

- LEO satellites are stationed between 160km and 2,000km in altitude
- High enough to stop them slowing down by “grazing” the denser parts of the earth’s ionosphere
- Not so high that they lose the radiation protection afforded by the Inner Van Allen belt.
- At a height of 550km, the minimum signal propagation delay to reach the satellite and back is 3.7ms, and at the horizon its 18ms
If you use a minimum angle of elevation of 25° then at an altitude of 550km each satellite spans a terrestrial footprint of no more than ~900Km radius, or 2M K².

At a minimum, a LEO satellite constellation needs 500 satellites to provide coverage of all parts of the earth’s surface.

For high quality coverage the constellation will need 6x-20x that number (or more!)
Starlink Constellation

- 4,276 in-service operational spacecraft, operating at an altitude of 550km

https://satellitemap.space/
Looking Up

Starlink tracks satellites with a minimum elevation of 25°.

There are between 30 – 50 visible Starlink satellites at any point on the surface between latitudes 56° North and South.

Each satellite traverses the visible aperture for a maximum of ~3 minutes.
Starlink Scheduling

• A satellite is assigned to a user terminal in 15 second time slots
• Tracking of a satellite (by phased array focussing) works across 11 degrees of arc per satellite in each 15 second slot
Starlink Scheduling

- Latency changes on each satellite switch
- If we take the minimum latency on each 15 second scheduling interval, we can expose the effects of the switching interval on latency
- Across the 15 second interval there will be a drift in latency according to the satellite’s track and the distance relative to the two earth points
- Other user traffic will also impact on latency, and also the effects of a large buffer in the user modem
Starlink Spot Beams

• Each spacecraft 2,000 MHz of spectrum for user downlink and splits it into 8x channels of 250 MHz each

• Each satellite has 3 downlink antennas and 1 uplink antennas, and each can do 8 beams x 2 polarizations, for a total of 48 beams down and 16 up.

“Unveiling Beamforming Strategies of Starlink LEO Satellites”
Starlink's reports

$ starlink-grpc-tools/dish_grpc_text.py -v status

id:                    ut01000000-0000000-005dd55
hardware_version:      rev3_proto2
software_version:      5a923943-5acb-4d05-ac58-dd93e72b7862.uterm.release
state:                 CONNECTED
uptime:                481674
snr:
seconds_to_first_nonempty_slot: 0.0
pop_ping_drop_rate: 0.0
downlink_throughput_bps: 16693.330078125
uplink_throughput_bps: 109127.3984375
pop_ping_latency_ms: 49.5
Alerts bit field: 0
fraction_obstructed: 0.04149007424712181
currently_obstructed: False
seconds_obstructed: 1.957997697103882
obstruction_duration: 540.0
direction_azimuth: -42.67951583862305
direction_elevation: 64.61225128173828
is_snr_above_noise_floor: True
Reported Capacity & Latency

![Graph showing reported capacity and latency over time. The left graph displays capacity in Mbps with time in seconds on the x-axis and capacity on the y-axis. The right graph shows latency in ms with time in seconds on the x-axis and latency on the y-axis.]
Reported Capacity & Latency
• This is going to present some interesting issues for conventional TCP
• TCP uses ACK pacing which means it attempts to optimize its sending rate over multiple RTT intervals
• The variation in latency and capacity occurs at high frequency, which means that TCP control is going to struggle to optimise
How well does all this work?

Speedtest measurements:

We should be able to get ~120Mbps out of a Starlink connection. Right?
Link Characteristics

Speedtest Latency:
Link Characteristics

1-second ping

Highly unstable jitter

Micro-drops

24 Hours
TCP Flow Control Algorithms

“Ideal” Flow behaviour for each protocol
iperf3 - cubic, 60 seconds

TCP with CUBIC

Loss

CUBIC
Qperf – quic (with cubic)
iperf3 - bbr

TCP with BBR

Download Rate (Mbps)

Seconds

Loss
Cubic, Quic/Cubic, BBR
Protocol Considerations

• Starlink services have two issues for transport protocols:
  • Very high jitter rates
  • High levels of micro-loss

• Loss-based flow control algorithms will over-react and pull back the sending rate
  • Short transactions work well
  • Paced connections (voice, zoom) tend to work well most of the time
  • Bulk data transfer not so much

• It’s better to use a conventional TCP control with a large SACK window or use loss-insensitive flow control algorithms, such as BBR, to get high transfer rate performance out of this service
Questions?