Deep Space IP

IETF 119, Brisbane, Australia

Marc Blanchet, marc.blanchet@viagenie.ca, March 2024

Agenda

- Administrativia
- What is Deep Space IP?
- Update on testbed
- COAP for deep space IP
- Congestion control discussion

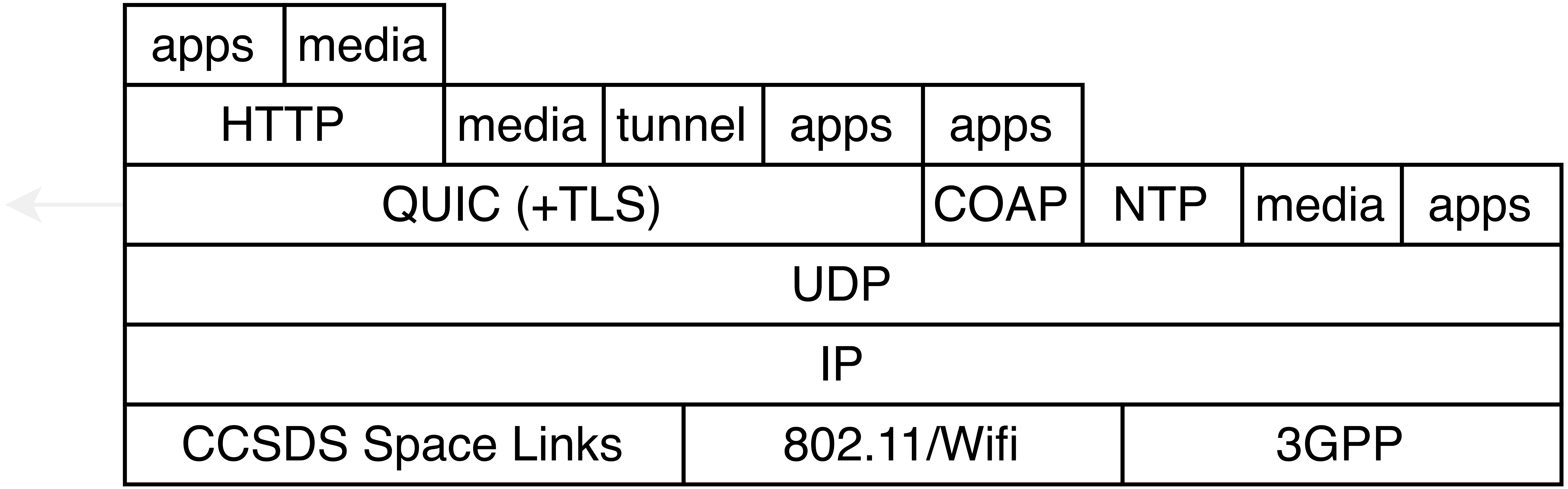
Administrativia

- Meeting under IETF Note Well
- Group Mailing list: deepspace@ietf.org
    - Subscribe at: https://www.ietf.org/mailman/listinfo/deepspace
- Group web site (using GitHub Pages):
    - https://deepspaceip.github.io/
    - Repo for slides, meeting notes, drafts, projects, issues, …
    - If interested in contributing, send me a note.
- QUIC in space Slack sub-channel under quicdev.slack.com main channel (send me a note if you want to join)
- This meeting remote access: https://ietf.webex.com/meet/sidemeetingietf1

What is Deep Space IP?

- Context: Deep space communications has specific characteristics, such as long delays and disruptions
- Goal: Using the Internet Protocol suite in deep space, as an alternative to the Bundle Protocol
- Work: Investigating how to profile the IP protocols and apps to make them work in deep space
- Key considerations:
    - IP forwarding: on an intermediate node, do not drop but instead store IP packets when there is no entry to destination in the forwarding table (same requirement as bundle storage in forwarders)
        - To handle intermittent connectivity
    - Transport profile (how to run QUIC in this context, but others too)
    - Routing
    - Applications and Applications protocols profiling (set larger timers…)

Deep Space IP Protocol Stack



- While initial focus has been on QUIC, COAP or native UDP protocols/apps are being investigated and profiled.

Update on Deep Space IP Testbed

Netem Update

- Netem
  - Linux TC Netem enables simulating various network conditions such as delay, packet loss, duplication, reordering, queueing, …
  - The delay maximum value has been limited to 274s. Ok for basic tests, but was limiting factor. For the testbed, we had to induce delays by other means which were more complicated.
  - After some email exchanges with iproute2/tc/netem author and maintainer, he provided a patch and we tested it since the last 2 weeks, works great! and the fix has been committed very recently to iproute2 repo. Commit. (Send me email if you need help to use it)
  - New maximum value is 2^64 seconds!

QUIC Stack

- The Quinn QUIC stack has been used recently for the testbed.
  - Written in Rust
  - Has an extensive API to set almost all possible transport parameters. Easier for prototyping.
  - Has an HTTP client and server implementation
  - Big thanks to Benjamin Saunders (co-author) for helping.

Updates to Quinn HTTP client and server

- Default transport parameters have been changed in the HTTP client and server:
    - Implements a base line « comparison » with Bundle Protocol (which has no congestion control)
    - Path MTU discovery almost disabled.
        - PMTUD is probably not that useful in deep space since all links are well known and managed and the number is small
        - It could be kept as optional depending on the needs, just creates a few additional packets
    - Set various parameters to large values: initial_rtt, max_idle_timeout, ack_frequency, window sizes
    - Implemented a simple congestion controller: only sets the window size
        - This enables pacing the data sent by the client, based on the mission needs
        - Still, QUIC manages reordering, loss, duplicates, …
    - Not yet implemented: NCID pool. May or may not be needed. Or a small one.
    - All these are settings passed as arguments on the cli at the moment of the request.
        - They can be adapted and used depending on the mission/application/use case

Early Results

- Done very recently (thanks to netem update)
- Work in Progress

An HTTP Request to Voyager!

- 18 hours (64800 seconds) delay each way; 36 hours RTT

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.65.33	192.168.65.25	QUIC	1242	Initial, DCID=d61b8e047f
2	64800.438656	192.168.65.25	192.168.65.33	QUIC	1380	Handshake, DCID=2f26ef8a
3	129600.8077…	192.168.65.33	192.168.65.25	QUIC	1242	Handshake, DCID=bf92a7a2
4	129600.8086…	192.168.65.33	192.168.65.25	QUIC	200	Protected Payload (KP0),
5	194401.1215…	192.168.65.25	192.168.65.33	QUIC	691	Protected Payload (KP0)
6	259201.4231…	192.168.65.33	192.168.65.25	QUIC	79	Protected Payload (KP0),
7	259201.4236…	192.168.65.33	192.168.65.25	QUIC	96	Protected Payload (KP0),
8	259201.4245…	192.168.65.33	192.168.65.25	QUIC	86	Protected Payload (KP0),

  - 1-2: client-server initial connection handshake. Crypto set.
  - 3. NCIDs. Haven’t yet worked on it.
  - 4. GET HTTP REQUEST
  - 5. HTTP RESPONSE
  - 6. NCIDs. Haven’t yet worked on it.
  - 7-8. Client connection close

An HTTP Request to Voyager!

- Since QUIC have streams, a connection between two peers can be set for loooong time and all HTTP requests/responses and other apps can use it.
- So the connection establishment (first 2) and the connection close (last 2) are done « once » maybe per hour/day/week/month/year to/from a spacecraft.
- The HTTP Request and response then takes only 1 packet sent and 1 packet received and 1 RTT
    - Or more if they cannot fit in a packet.
    - Forgetting NCIDs (that may or may not be « fixed »)

Other Tests Done

- Repeat X times the same HTTP request within the same connection.
    - Demonstrate the use of an established QUIC connection (lower total time).
    - (Same request just because it is simpler to test, verify and automate).
- Reorder:
    - early test of 30% reorder with HTTP request 10x repeat succeeded (all 10 responses received) and show a few more packets added. More analysis to do.
-

One more thing…

Network Management

- SNMP uses UDP. No timeout defined in the SNMP protocol, nor UDP.
  - Timeout is set by the client.
- Test:
  - net-snmp on client and server
  - Delay of 2h each way = 4h RTT
  - snmpget -t 14500 $oid   (14400 = 4 hours)
  - Just worked!
- Network Management: ✅

Time Distribution

- NTP uses UDP.
- NTP is a complex time machinery to compensate network conditions to accurately set time based on remote reference servers.
- NTP experts were consulted, and they are pretty sure that NTP would work with looong delays, except that the notorious precision (< 10ms) will not be as good. Ok!
- Let's put it to test!
    - ntpd on server, serving its own time
    - ntpdate on client
    - Delay of 2h each way = 4h RTT
    - Set artificially the client system date to a very bad value outside of the RTT. Aka 10 hours behind. (pretty bad clock drift!)
    - sudo ntpdate -t 14500 $serverIPaddress
    - Worked. Accuracy was ~30 seconds
    - Do it second time: sudo ntpdate …
    - Worked. Accuracy was ~2 seconds
- Time Distribution: ✅

Testbed Future Work

- QUIC related:
    - Need to investigate the NCID additional packet (recipe known, need to be implemented)
    - Use the key log file to decrypt and analyze
    - Packet loss, duplicate, reorder, …
    - Migration
    - More analysis and tuning of parameters
- More automation
    - Automate everything based on scenarios described in a JSON file
    - Automate graphics and tabular results (like QLOG flows)
- Time warp: QUIC or Linux level.
- Additional protocols and applications
- Complex network with disruptive events
- Publish results