ROS2 Over Constrained Links

Charles Cross
Teleoperations Lead

voyage

Topics

Intro

Message Design & QOS Considerations

Creating Boundaries

Custom Transports

Teleoperation over LTE

Q&A

voyage

Scope of this discussion

voyage

Mainly addressing

- Lossy and/or bandwidth-limited mediums (<30 Mbps)
  - Wireless networks
    - **WiFi**
    - **Cellular**
    - Bluetooth
    - IEEE 802.15.4 based protocols
    - Custom radio links
    - And more...
  - Communication over public WANs
  - Physical links (point-to-point or arbitrated bus topologies)
    - RS232/422/485
    - High-speed CAN (CAN-FD)
    - Ethernet in bus topology (no switch)
  - Other non-IP communication interfaces
- Situations where latency can be critical
  - Real-time video, audio, and control

Not addressing

- Generally reliable, high-speed interfaces
  - T10/100/1000 Ethernet based LANs
- Resource constrained devices
  - I.e. microcontrollers
  - DDS-XRCE standard addresses this
  - See:
    - Micro-ROS/ROS2 Embedded SIG
    - eProsima Micro-XRCE-DDS
    - PX4's RTPS/ROS2 Interface

Message Design Considerations

First, let's examine the standard shapes demo type:

Name	Type / Ordinal	Extensibility	Optional?	Min Sample Size	Max Sample Size	TypeCode Size	TypeObject Size
ShapeTypeExtended	ShapeTypeExtended	extensible		28	160	378	1,060
{base type}	ShapeType	extensible		20	148	130	552
color (key)	string<128>	mutable	false	5	133	12	196
x	long	final	false	4	4	6	36
y	long	final	false	4	4	6	36
shapesize	long	final	false	4	4	6	36
fillKind	ShapeFillKind	extensible	false	4	4	156	284
SOLID_FILL	0						
TRANSPARENT_FILL	1						
HORIZONTAL_HATCH_FILL	2						
VERTICAL_HATCH_FILL	3						
angle	float	final	false	4	4	6	36

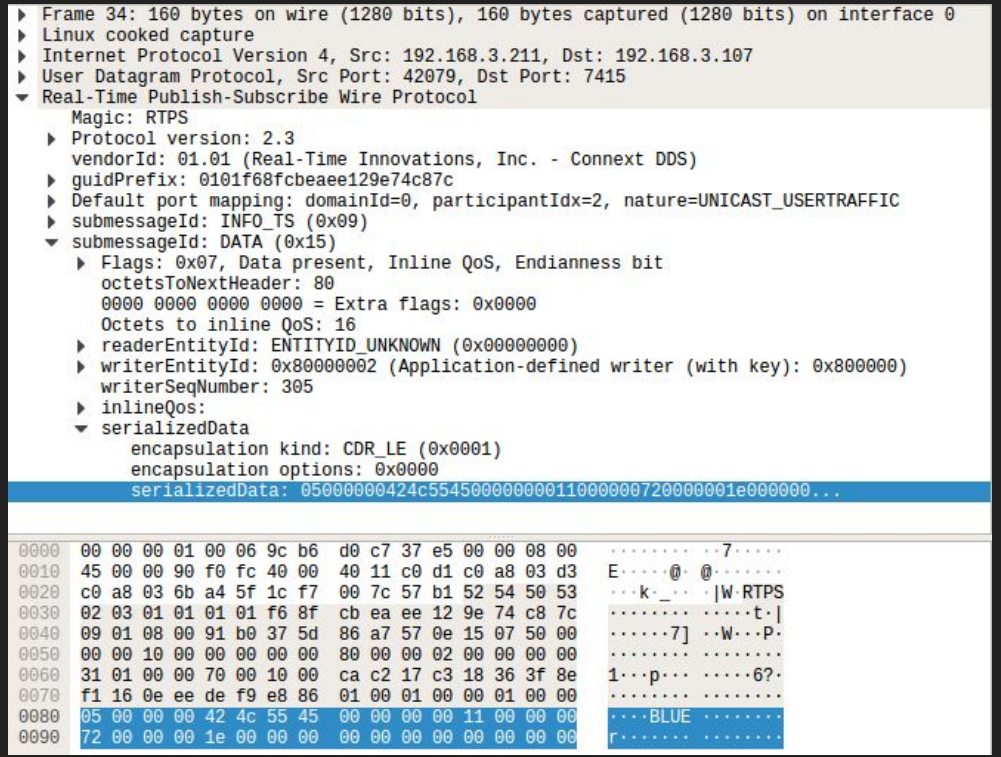
Message Design Considerations

voyage

For a single sample...

- Ethernet Frame (14B header + 4B FCS)
- IPv4 Header (20B)
- UDP Header (8B)
- RTPS Message (varies, 116B here)
  - Header info (20B)
  - INFO_TS submessage (12B)
  - DATA submessage (84B)
    - Actual payload (32B)

At the end of the day, for a 32 byte message, we had to send 162 bytes over the wire.



Message Design Considerations

Some observations:

- Overall, for such a small message, we spent 400% overhead on protocol headers
- RTPS appears to add significant overhead to our communication
  - Some of this information is useful or necessary in our application (INFO_TS submessage)
  - Other parts of the header remain relatively static across messages, oftentimes
  - Much of this information is used by the middleware to ensure proper delivery within large distributed systems
    - For some use-cases, a good portion of it may not be necessary or can be inferred
- Sending at 50Hz, total bandwidth usage is 64.8Kbps
  - Assuming best-effort delivery
  - Reliability will add additional overhead
  - Doesn’t seem like much, but it stacks up with each data stream you create!
    - Don’t forget discovery, liveliness, and reliability protocol overhead!

Message Design Considerations (small data)

voyage

Recommendation: When operating over constrained links, try to keep your messages small, but more importantly, try and consolidate/aggregate or filter datastreams as much as possible.

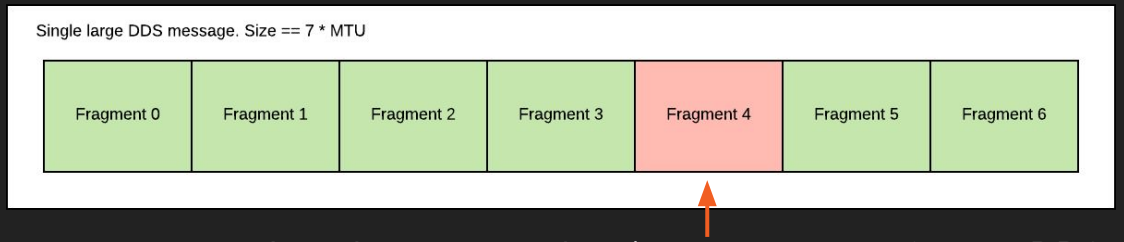
- Aggregation:
  - Ex. if you have three separate topics describing the state of a robot where each message consists of a single Float, try to aggregate those into a single datatype before sending to the remote system.
  - This will save you significant overhead, particularly if the state is updated frequently.
    - One message that is roughly: (OVERHEAD + DATA_SIZE * 3)
    - ...vs three messages that are: (3 * (OVERHEAD + DATA_SIZE))
  - Additional savings on discovery and metatraffic!
    - ...especially if you have nodes coming up and down a lot
- Filtering:
  - Consider that for many use-cases, your system may be producing data at much higher rates for internal purposes than remote systems require
  - Ex. IMU data being published at 400Hz being consumed by a kalman filter block locally and a teleoperator remotely.
    - Most often, data rates of 1-60Hz suffice for the remote operator or GUI systems
  - For use-cases where you need all of the data (i.e. file transfer), consider creating intermediate types that batch multiple samples

Message Design Considerations (large data)

For the purposes of this discussion, I will consider “large data” to refer to messages that span multiple MTU-sized frames. Typical use-cases include video streaming and file transfer.

Considerations:

- By default, DDS samples will generally be able to exceed your system’s MTU size
- When your OS fragments a UDP packet, all fragments must be received on the remote side to reassemble the message.
  - Ex. H264/HEVC encoded video data will often range from 1KB to 100KB+ per frame, triggering this
  - If any fragment is lost, the whole sample is lost
  - For best-effort delivery, game over
  - For reliable delivery, DDS will need to retransmit **the entire sample**
    - Solution for this coming on the next slide



Message Design Considerations (large data)

voyage

Things to consider:

- Be aware of the fragmentation behaviors present in your network
- Consider tuning retransmission properties in network equipment (if possible)
- DDS Participant Transport QOS feature: Max Message Size
  - Allows the middleware to perform fragmentation of DDS messages
    - If you set this to be at or under the MTU size, you can avoid the issue where loss of a single fragment in a reliable topic causes retransmission of multiple UDP fragments
    - The middleware can keep track of which fragments need to be retransmitted
    - **NOTE:** Each middleware-fragmented packet will have additional RTPS overhead
    - You can sometimes tune this based on the loss characteristics of the network
  - Often set up to default to 65507 (max UDP packet size - IPv4 header - UDP header)
  - Still doesn't help us when using Best-Effort: the sample is unable to be reassembled either way
- Alternatively, consider fragmenting larger message types yourself
  - Pro:
    - Flexibility in how to handle reliability, retransmission, and error correction based on use-case
  - Cons:
    - A bit complicated to calculate how to design your messages to achieve the most efficient size
    - May vary across different local/remote systems, networks, and network interfaces
    - Reassembly and filtering of certain types of redundant samples is now your job

QOS Considerations

Understand how various QOS parameters affect network usage and ‘real-time’ behavior in latency sensitive applications. Some relevant policies and parameters:

- Participant
  - Transport properties
    - Max message size
    - Which transports are available to be used
    - Which network interfaces are whitelisted/blacklisted
    - Multicast enable/disable
  - Discovery configuration
    - Announcement configuration
    - Liveliness lease duration & assert period
- Datareaders/Datawriters
  - Reliability protocol configuration
    - Heartbeat periods and behaviors
    - ACK/NACK timing parameters
    - Disabling positive acknowledgment
      - Min/Max sample keep duration
    - Repair characteristics
- And more...

QOS Considerations (caveats and further reading)

voyage

It is important to mention that use of some of these QOS features is limited due to:

- Differences in DDS implementations across vendors
  - Some are only implemented/exposed by RTI as extensions to the DDS standard
  - FastRTPS should have most of what we have talked about so far, but no guarantees

- ROS2, taking a more middleware-agnostic stance, makes an attempt at choosing sane default values for many of these lower-level configurations based on the needs of general users and does not necessarily make access to some of these QOS policies readily available
  - Many need to be specified at creation/enable time of the related DDS entities
    - Participant Transport properties
    - Datareader/Datawriter reliability protocol
  - You can always work around this by creating a separate pure DDS/RTPS application that acts as a bridge
  - Individual DDS implementations often provide a means of setting some default QOS via env variables and files

- Bounds around bandwidth utilization and message timing in DDS are predictable and tunable, but require a fine-grained understanding of these policies and the standard
  - RTI's User Manuals are a great resource, providing detailed timing diagrams for most behaviors & protocols
  - As your ROS2/DDS databuses grow beyond a few participants/topics, Wireshark gets harder to grok
    - Scripting libraries like Scapy and Pyshark can help automate performing analysis of RTPS traffic

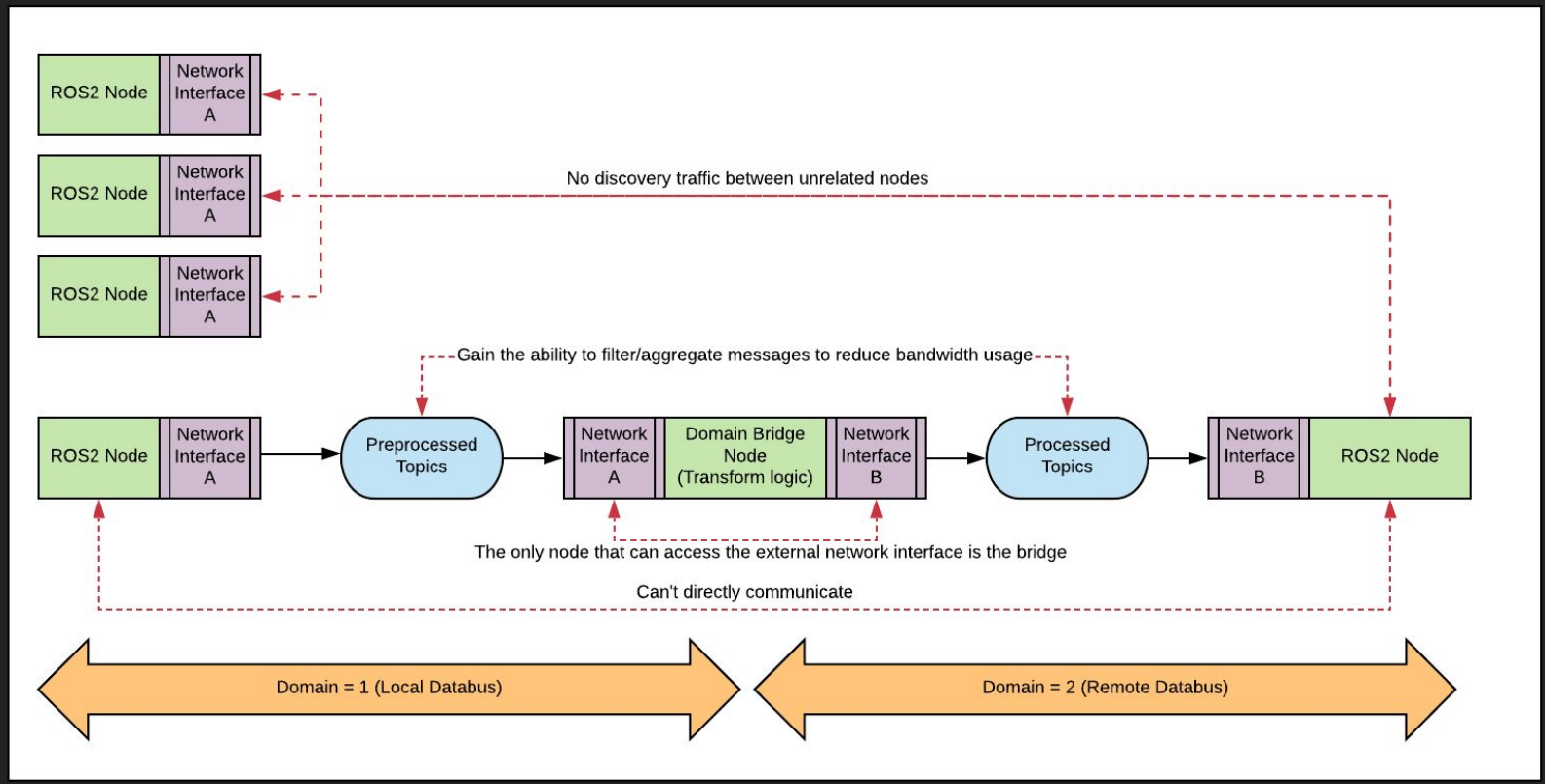
Creating Boundaries

voyage

- At a high level, we should strive to make sure only relevant data is transmitted over constrained links
  - Cut down on discovery traffic overhead
  - Reduce/eliminate the chance of unintentionally publishing/subscribing across certain interfaces
  - Control which network interfaces/transports are available to different groups of processes
  - Perform filtering/aggregation at these boundaries

- DDS has a few mechanisms for helping with this:
  - Domain Participant Transport QOS Policies
    - Allow you to configure which transports and network interfaces are available to a participant
  - Domains
    - Participants in different domains will not communicate with each other at all
    - Some discovery traffic may be exchanged but discarded, depending on use of multicast & initial peers
    - Fixed at creation time
  - Partitions
    - Applied at Publisher/Subscriber level (domain participants will still exchange discovery metatraffic)
    - Publishers and Subscribers that don’t match partitions will not trade discovery info on writers/readers
    - Can be dynamically changed at runtime
  - Security Protocol
    - Highly configurable access control and encryption can prevent discovery/communication at multiple levels

Creating Boundaries



voyage

Custom Transports (and more)

- We have examined at a high level many of the items that should be taken into consideration when trying run ROS2/DDS over lossy, constrained networks
  - Most problems have multiple solutions with different pros/cons, depending on use-case and requirements
  - May be hard to understand how to use all of the various pieces to tackle concrete use-cases so...

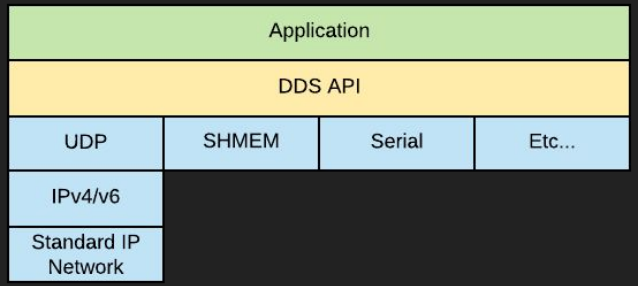
Let’s examine a use-case that we are tackling at Voyage:

Real-time teleoperation of vehicles over LTE

Custom Transports (and more)

voyage

- Before enumerating the challenges posed by this use-case, let's take a look at a feature present in most DDS implementations that will be important: Custom, Pluggable Transports
  - At a high-level, this feature allow us to write plugins in the form of libraries that get loaded by the middleware implementation which are passed the final RTPS payloads to be sent across whatever 'wire' the plugin manages.
  - All DDS vendors provide a built-in UDP Transport
  - Some provide a shared memory transport
  - You can specify which transports are allowed on a per datawriter basis
  - Allows specification of a 'transport priority' QOS integer value on a per-datawriter or per-topic basis
  - Supports a parent-child hierarchy - you can link multiple plugins together
  - Enables the use of non-IP centric transports, as you can pass the RTPS data through any link



Custom Transports (and more)

voyage

- So what problems does this feature help to solve, generally?
  - Enabling the creation of custom interfaces to pass DDS traffic through
    - Constrained network mediums
    - .... Or using UDP/TCP to create opaque, encrypted tunnels across untrusted networks
  - Retaining the ability to allow DDS to handle reassembly, reliability, redundant data filtering, etc.
  - Remember the overhead of the RTPS headers and traffic from earlier?
    - With transport plugins, you can add header and data compression (where sensible) to save some bytes
    - You can also customize how discovery is done
    - See RTI's docs on their Limited Bandwidth Plugins:
      - https://community.rti.com/static/documentation/connext-dds/6.0.0/doc/manuals/connext_dds/limited_bandwidth_plugins/RTI_LimitedBandwidthPlugins_UsersManual.pdf
  - If using UDP/TCP as a transport mechanism, you have control over how many/which ports to use and over what network interfaces (whether virtual or physical)
    - This can be useful when interfacing with thirdparty APIs, trying to simplify port forwarding configurations, or trying to perform NAT traversal and tunnel DDS traffic across public networks

Real-time teleoperation over LTE

voyage

- Why LTE?
  - WiFi is not well suited to the task:
    - High-tail latencies
    - Less predictable scheduling
    - Highly susceptible to interference on unlicensed bands
    - Real-time handover across networks not well supported
    - Covering and maintaining large geographical distributions (>30sqmi) is expensive and complex
    - Some newer IEEE standards have begun to improve upon all of the above, but not widely implemented
  - LTE protocols improve on all of the above
  - There is extensive existing Public LTE infrastructure, which for near-range operations (<100mi) often yields latencies between 40-80ms, suitable for real-time teleoperation under certain conditions

- Challenges
  - Carriers are in charge of scheduling and other network behaviors (throttling, typically on uplink)
  - Performance varies widely, depending on user density, capacity of infrastructure, time of day, etc. (congestion)
  - There are always gaps in coverage (and it is impractical to expect mobile carriers to stand up additional coverage)
  - Data usage can get expensive, depending on how often you are operating high bandwidth links
  - Though generally acceptable on average (assuming adequate infrastructure), latency, jitter, and packet loss rates are still dynamic and can exceed the safe limits for teleoperating a vehicle
  - Have to deal with NAT traversal and carrier-grade NAT

Real-time teleoperation over LTE

voyage

So how do we address some of these challenges?

- Changing connectivity, latency, jitter, and packet loss (from various sources)
  - First, and most effective strategy is to bond multiple LTE modems together
    - Allows:
      - Maintaining connectivity during handover on any individual modem
      - Redundantly sending packets across different modems to increase likelihood of receipt
      - Multiplexing of packets across different modems to increase effective throughput
      - Diversity of carriers, making complete interruption of service less likely
      - Diversity of spectrum used by carriers can increase robustness of overall connectivity
      - Use of multiple antennas placed around the vehicle increases transmit spatial diversity
      - Overall greater coverage
  - Second, for higher bandwidth, low-latency streams like video, we use forward error correction (FEC) to mitigate packet loss
    - Basic approach is to spend some additional overhead to include enough information to recover individual lost packets as an individual frame is being received
    - Relying on the DDS reliability protocol requires a round trip NACK+repair pass, impacting latency
    - By characterizing loss patterns, we can dynamically change the FEC strategy (including reducing usage)

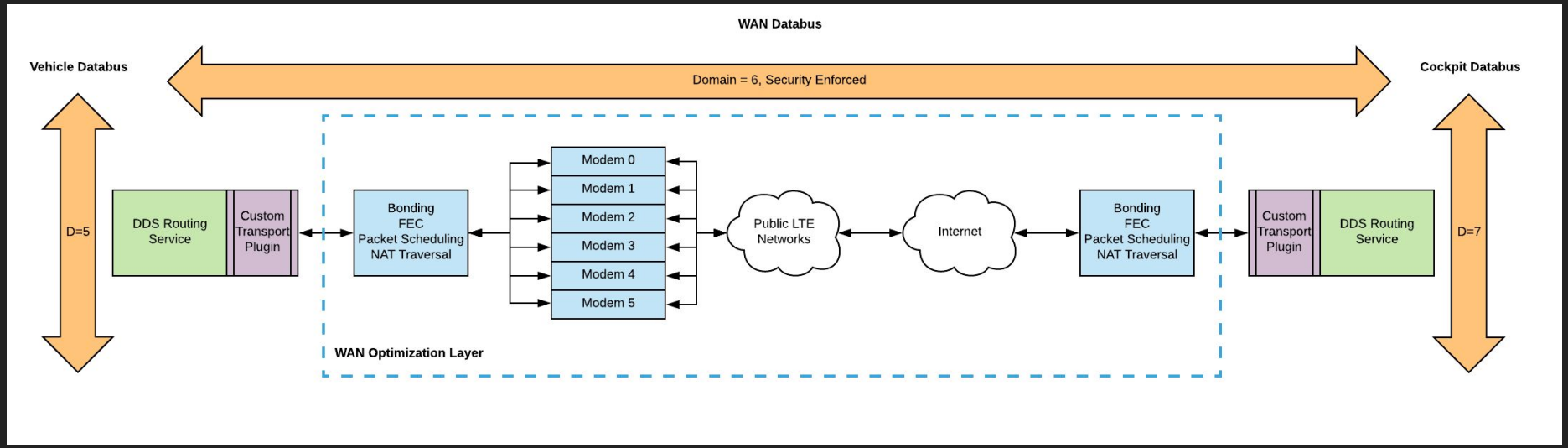
Real-time teleoperation over LTE

voyage

- Congestion
  - It is possible to characterize the effective throughput, latency, and loss patterns across each modem
  - With this information, it is possible to dynamically decide how to route individual packets to each modem
    - For heartbeats, status info, and control signals (usually small), it is effective to duplicate across all modems
    - For video and higher bandwidth streams, a combination of duplication and multiplexing can be used
    - With a custom transport, it is possible to use the ‘transport priority’ field to categorize these different streams of data.
      - Each stream can utilize a different packet scheduling/buffering/routing/FEC algorithm
  - Additionally, this information can be aggregated to provide feedback to the video/audio encoding pipeline
    - Dynamic adjustment of resolution, bitrate, framerate, optional sources, etc
    - There are many additional techniques pertaining to optimization of real-time video streams, in terms of how you can use network feedback to adjust the encoding process, a topic deserving of its own larger presentation.

- Security
  - Using DDS security, we can make sure that all data traversing this ‘WAN optimization layer’ is encrypted
  - Useful for leveraging third party bonding/network solutions that you may not be able to trust
    - Important if working in a sensitive, or safety critical context

Tying it all together...



- Vehicle & Cockpit traffic separated via domains
- Security enforced at multiple levels in each domain
- Routing Service blocks filter/aggregate datastreams
- Custom transport plugin passes DDS data to WAN service

- WAN service handles
  - Bonding across multiple modems
  - FEC
  - NAT Traversal
  - Provides feedback to encoding process (not shown)

Q&A

Thank you!

charles@voyage.auto

voyage