



Initial Implementation of a NetworkManager class

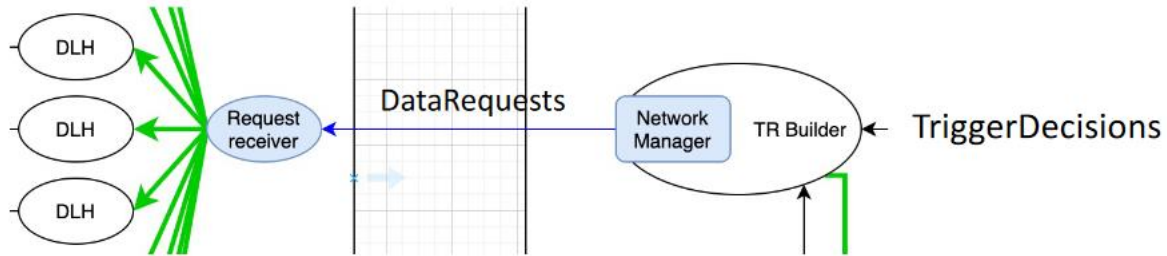
Eric Flumerfelt

DUNE DAQ Dataflow

28 September 2021

NetworkManager

- As Kurt detailed at the Collaboration Meeting last week, we have decided to move to a more direct communication model for some of the messages between DAQ applications
- The goal is for DAQModules to have the ability to communicate more directly over the network for cases where the queue infrastructure is not needed and/or to simplify configuration



NetworkManager Basic API

- The proposed API for the NetworkManager provides a callback-based interface for reception of messages. It also handles connection names and does name resolution and/or service discovery to get the correct endpoint configuration
- It provides a simple, IPM-like interface for sending messages, again taking a connection name and translating it to endpoint configuration













NetworkManager
<pre>+get(): NetworkManager& +start_listening(connection_name:std::string const&, callback:std::function<void(Receiver::Response)>): void +stop_listening(connection_name:std::string const&): void +add_subscriber(connection_name:std::string const&, topic:std::string const&, callback:std::function<void(Receiver::Response)>): void +remove_subscriber(connection_name:std::string const&, topic:std::string const&) +send_to(connection_name:std::string const&, buffer:const void*, size:size_t, topic:std::string const&=""): void const</pre>

NetworkManager Initial Implementation

- I have created an initial implementation of the NetworkManager API
- <https://github.com/eflumerf/networkmanager.git>
- Depends on *ipm* and *toolbox* (upcoming package to contain generic utilities, including DNS-resolving code)
- Test-driven development was used to implement API
- Currently uses static configuration for name translation

LCOV - code coverage report

Current view: top level	Hit	Total	Coverage
Test: dunedaq.info.cleaned	Lines: 1378	1415	97.4 %
Date: 2021-09-28 15:22:29	Functions: 370	379	97.6 %

Directory	Line Coverage ↕	Functions ↕
ipm/include/ipm	 100.0 % 21 / 21	85.7 % 12 / 14
ipm/plugins	 82.7 % 124 / 150	90.4 % 47 / 52
ipm/src	 100.0 % 16 / 16	100.0 % 2 / 2
ipm/unittest	 100.0 % 206 / 206	100.0 % 62 / 62
networkmanager/include/networkmanager	 100.0 % 6 / 6	100.0 % 6 / 6
networkmanager/src	 99.1 % 217 / 219	94.9 % 37 / 39
networkmanager/unittest	 100.0 % 259 / 259	100.0 % 69 / 69
toolbox/include/toolbox	 93.7 % 134 / 143	100.0 % 25 / 25
toolbox/include/toolbox/detail	 100.0 % 49 / 49	100.0 % 10 / 10
toolbox/src	 100.0 % 52 / 52	100.0 % 7 / 7
toolbox/test/apps	 100.0 % 48 / 48	100.0 % 31 / 31
toolbox/unittest	 100.0 % 246 / 246	100.0 % 62 / 62

Generated by: [LCOV version 1.15](#)

NetworkManager Implementation API

networkmanager

NetworkManager

```
-s instance: std::unique_ptr<NetworkManager>
-m_connection_map: std::unordered_map<std::string, Connection>
-m_receiver_plugins: std::unordered_map<std::string, std::shared_ptr<ipm::Receiver>>
-m_sender_plugins: std::unordered_map<std::string, std::shared_ptr<ipm::Sender>>
-m_registered_listeners: std::unordered_map<std::string, Listener>
-m_registered_subscribers: std::unordered_map<std::string, Subscriber>
+get(): NetworkManager&
+start_listening(connection_name:std::string const&,
                 callback:std::function<void(Receiver::Response)>): void
+stop_listening(connection_name:std::string const&): void
+add_subscriber(connection_name:std::string const&,
               topic:std::string const&,
               callback:std::function<void(Receiver::Response)>): void
+remove_subscriber(connection_name:std::string const&,
                  topic:std::string const&): void
+send_to(connection_name:std::string const&,
         buffer:const void*,size:size_t,topic:std::string const&=""): void const
+receive_from(connection_name:std::string const&,
              timeout:Receiver::duration_t): Receiver::Response const
+configure(connections:std::unordered_map<std::string,
std::string>): void
+reset(): void
+get_connection_string(connection_name:std::string): std::string const
+is_connection_open(connection_name:std::string const&,
                    direction:ConnectionDirection=ConnectionDirection::Recv): bool const
+is_listening(connection_name:std::string const&): bool const
+has_subscriber(connection_name:std::string const&,
                topic:std::string const&): bool const
-NetworkManager()
-create_receiver(connection_name:std::string const&): void
-create_sender(connection_name:std::string const&): void
```

<<Enumeration>>

NetworkManager::ConnectionDirection

Recv
Send

Listener

```
-m_connection_name: std::string
-m_callback: std::function<void(Receiver::Response)>
-m_listener_thread: std::unique_ptr<std::thread> = nullptr
-m_is_listening: std::atomic<bool> = false
+Listener(connection_name:std::string const&)
+start_listening(callback:std::function<void(Receiver::Response)>): void
+stop_listening(): void
+shutdown(): void
+is_listening(): bool const
-startup(): void
-listener_thread_loop(): void
```

Subscriber

```
-m_connection_name: std::string
-m_callbacks: std::unordered_map<std::string, std::function<void(Receiver::Response)>>
-m_subscriber_thread: std::unique_ptr<std::thread> = nullptr
-m_is_running: std::atomic<bool> = false
+Subscriber(connection_name:std::string const&)
+add_callback(callback:std::function<void(Receiver::Response)>,
              topic:std::string const&=""): void
+remove_callback(topic:std::string const&=""): void
+has_callback(topic:std::string const&): bool const
+num_callbacks(): size_t const
+topics(): std::unordered_set<std::string> const
+is_running(): bool const
+shutdown(): void
-startup(): void
-subscriber_thread_loop(): void
```

Notes/Thoughts

- As Brett has noted, ZMQ sockets are not thread-safe, which means that I was unable to conform to Giovanna's suggested API where connections were opened and then listened to as separate steps
- Some form of standardized naming class should be created that is accessible to DAQModules to generate connection names as well as statically within a CCM context. (e.g. for configuring K8s socket entries)
 - Input destination information (i.e. GeoID), message type
 - Output connection name "Part3:DataRequest:RU-55" (May be modified to become a valid DNS name)
- The current implementation is completely static; dynamic lookup needs to be added and test applications written for a K8s environment