SBN and Protobetter Design Docs

By Danrae Pray

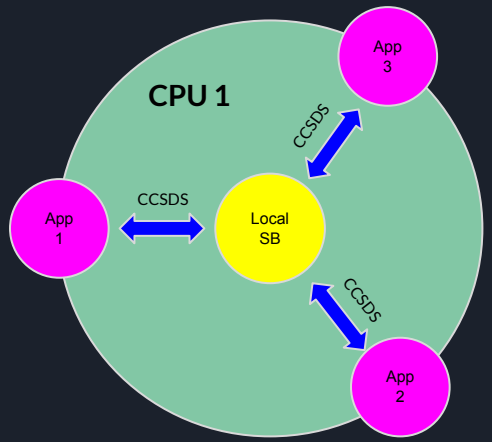
Contents

I) Introduction

SBN-UDP is an **unreliable**, **connection-oriented, application-layer networking protocol** that extends the CFS 'Software Bus' publish-subscribe pattern for communication between CFS applications across network interfaces.

At the time of this writing, it is implemented in a Core Flight Software (CFS) application called the 'Software Bus Network', or simply 'SBN'.

II) The 'Local Software Bus'



In NASA's Core Flight Software framework, applications should not access the memory of other applications directly. Rather, they should communicate by passing CCSDS messages through the 'Software Bus' API. The CFS software bus is generally referred to as the 'Local Software Bus' or simply 'Local SB'.

II) The 'Local Software Bus'

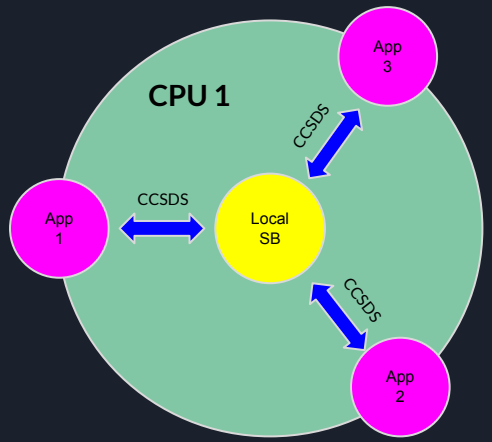
A CCSDS message has a standardized header followed by a user-defined payload (more info on CCSDS can be found here: https://public.ccsds.org/default.aspx).

An important feature of the software bus is that CFS applications which are consumers of data are decoupled from producers of the data through this local Software Bus API. Applications don't subscribe to data from a particular source directly. Rather, they publish & subscribe to CCSDS data via the Software Bus API based on a particular message ID encoded in the message header.

II) The 'Local Software Bus'

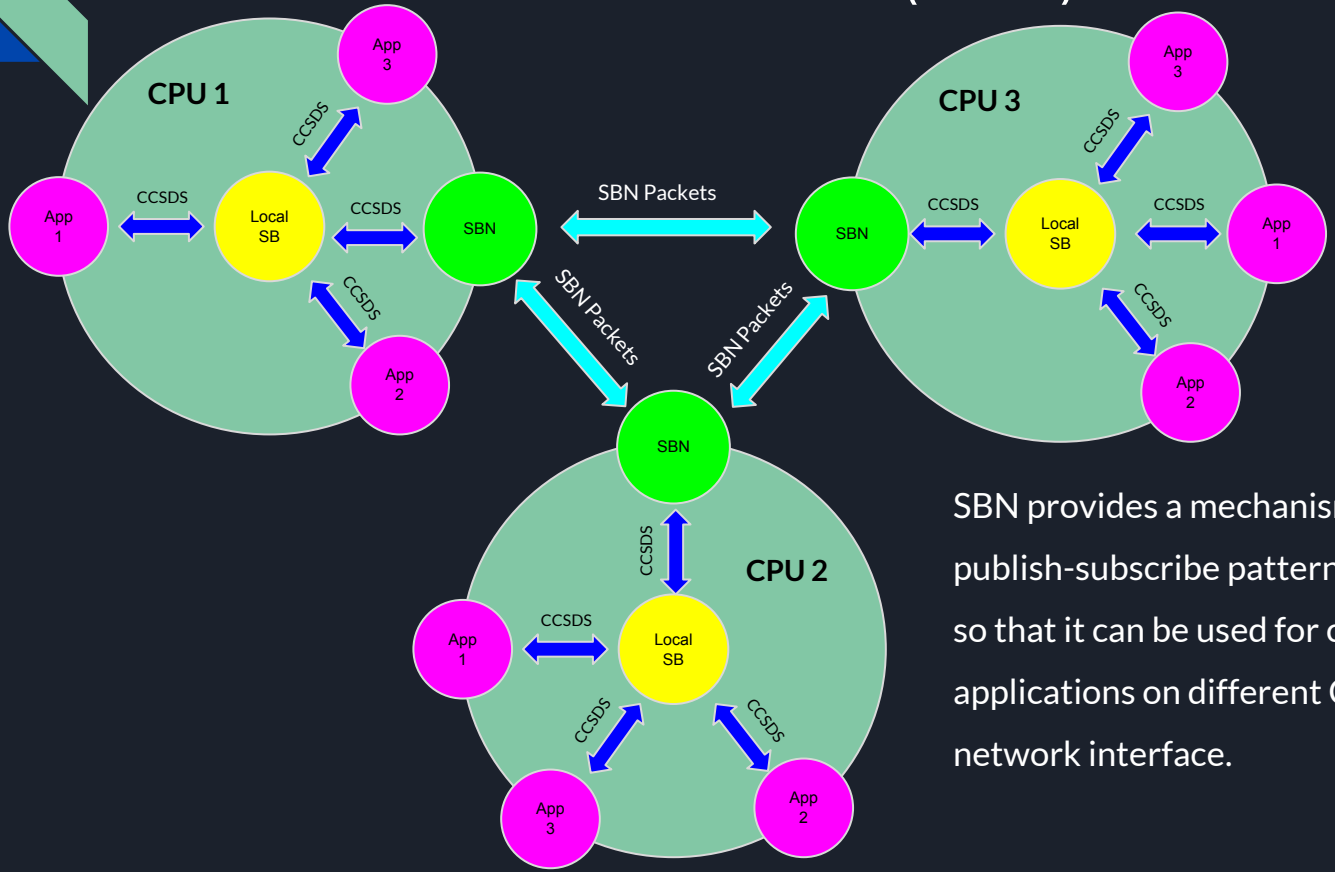
On a POSIX-compliant OS, a CFS instance (or 'CPU') may consist of multiple apps each running on their own threads within a single process where the actual inter-thread communication in the local SB is done via the POSIX message queue API. On a real-time OS such as VxWorks, the apps run as separate real-time processes communicating via the VxWorks APIs.

III) Distributed CFS Applications and the Software Bus Network (SBN)



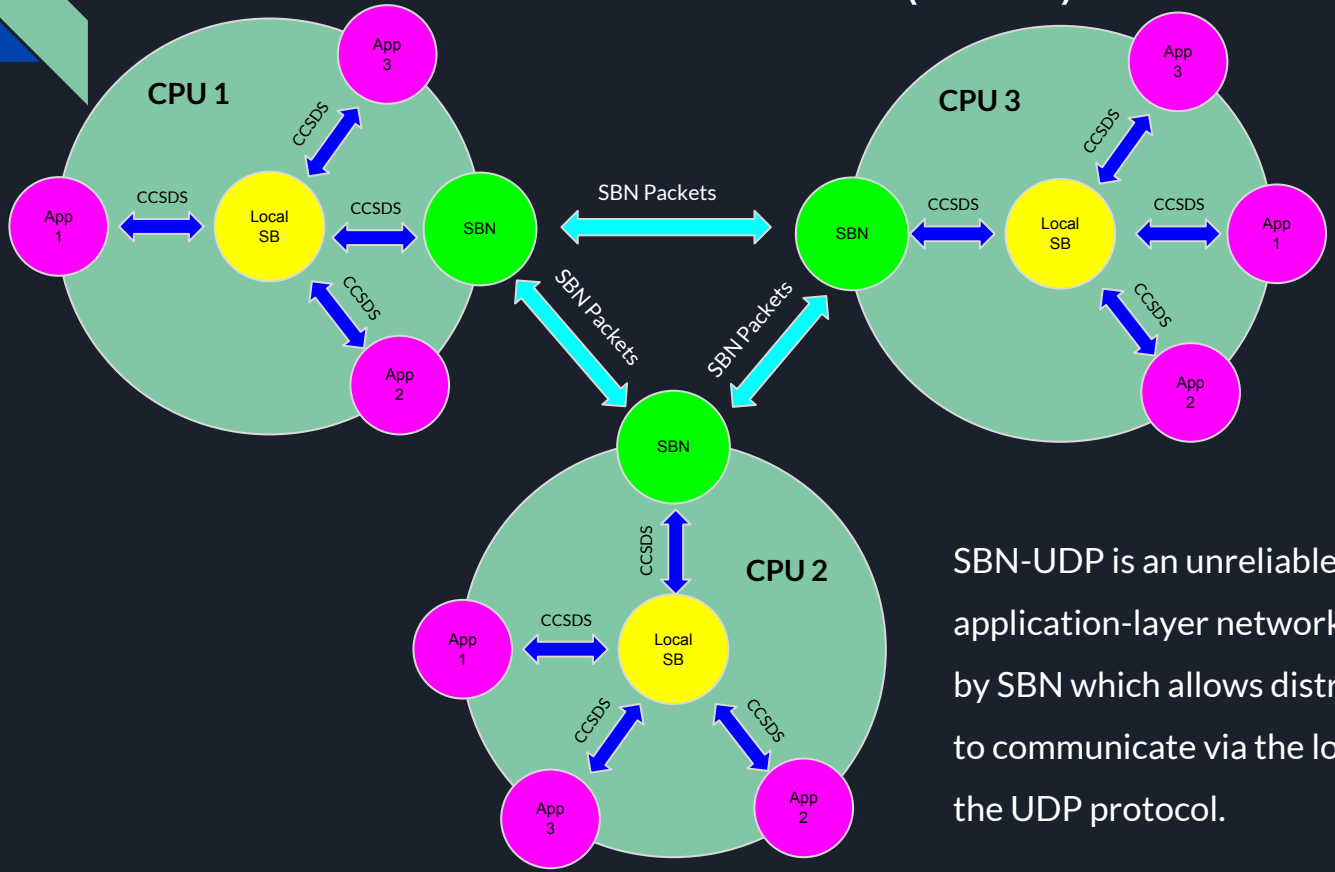
The local software bus is fine for multiple applications running on the same CPU. However, it has no way of communicating messages to other CFS instances, particularly ones on a separate CPU. This is where the Software Bus Network CFS Application comes into the picture.

III) Distributed CFS Applications and the Software Bus Network (SBN)



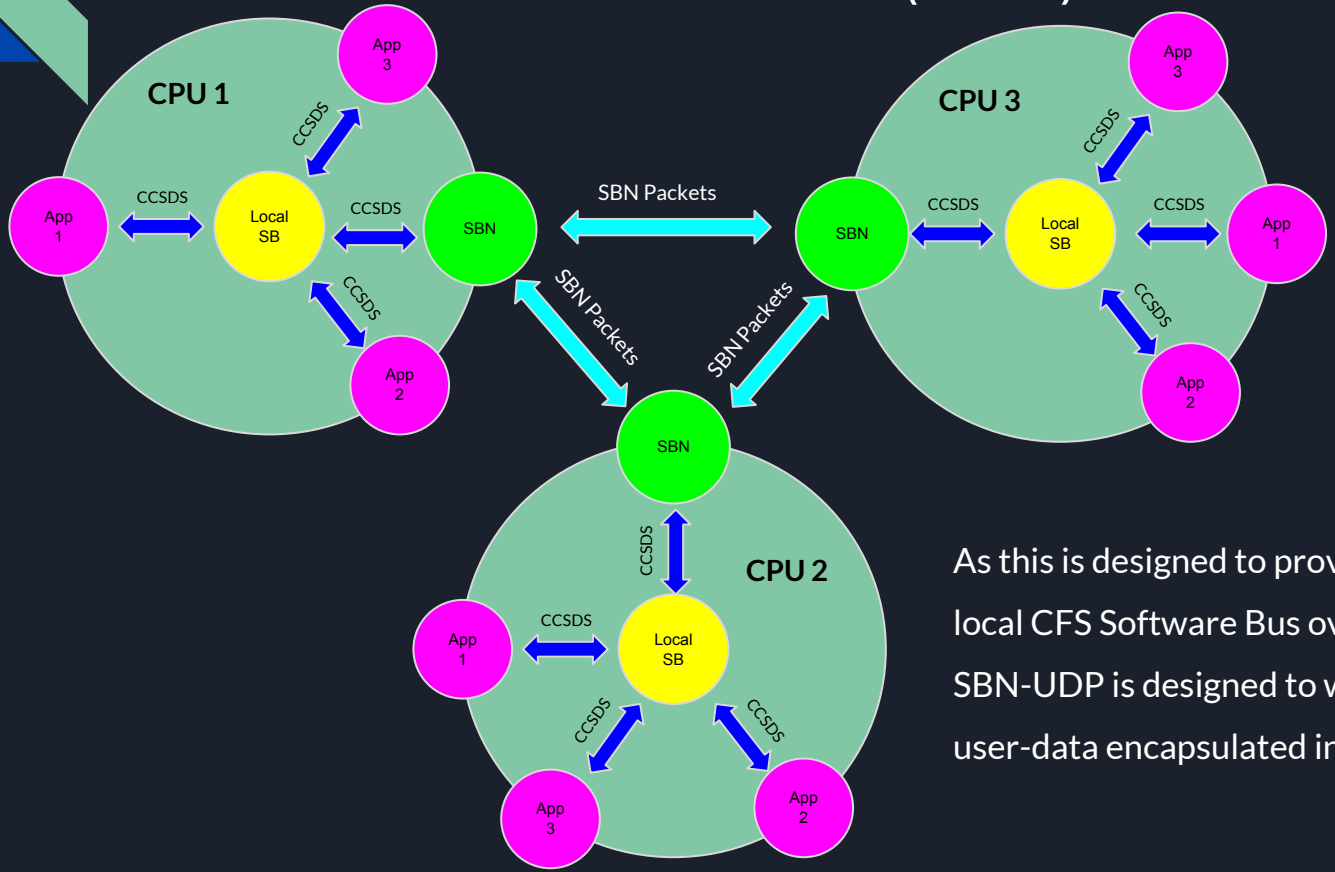
SBN provides a mechanism for extending the publish-subscribe pattern of the local Software Bus so that it can be used for communication between applications on different CPUs transparently via a network interface.

III) Distributed CFS Applications and the Software Bus Network (SBN)



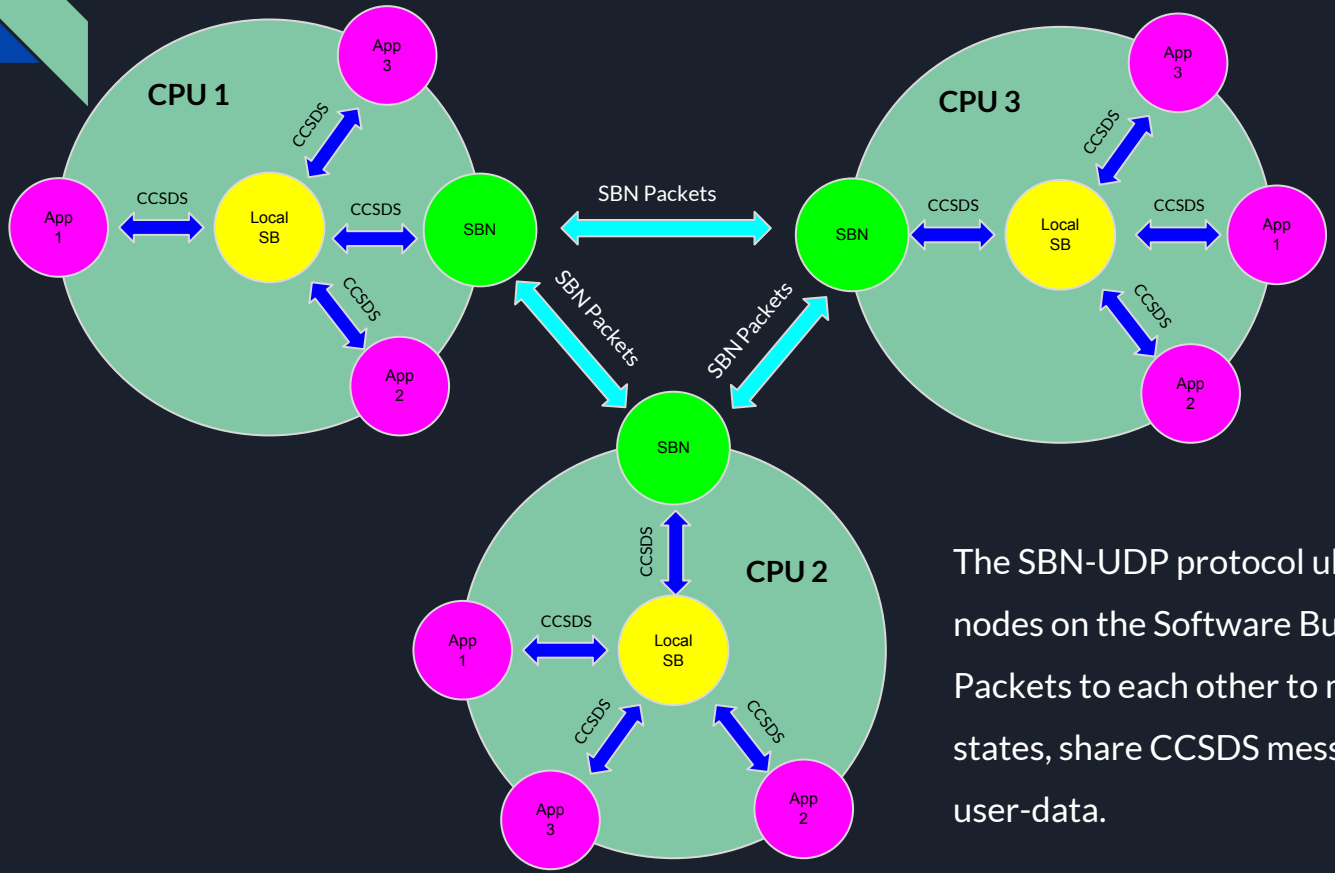
SBN-UDP is an unreliable, connection-oriented application-layer networking protocol implemented by SBN which allows distributed CFS applications to communicate via the local Software Bus API over the UDP protocol.

III) Distributed CFS Applications and the Software Bus Network (SBN)



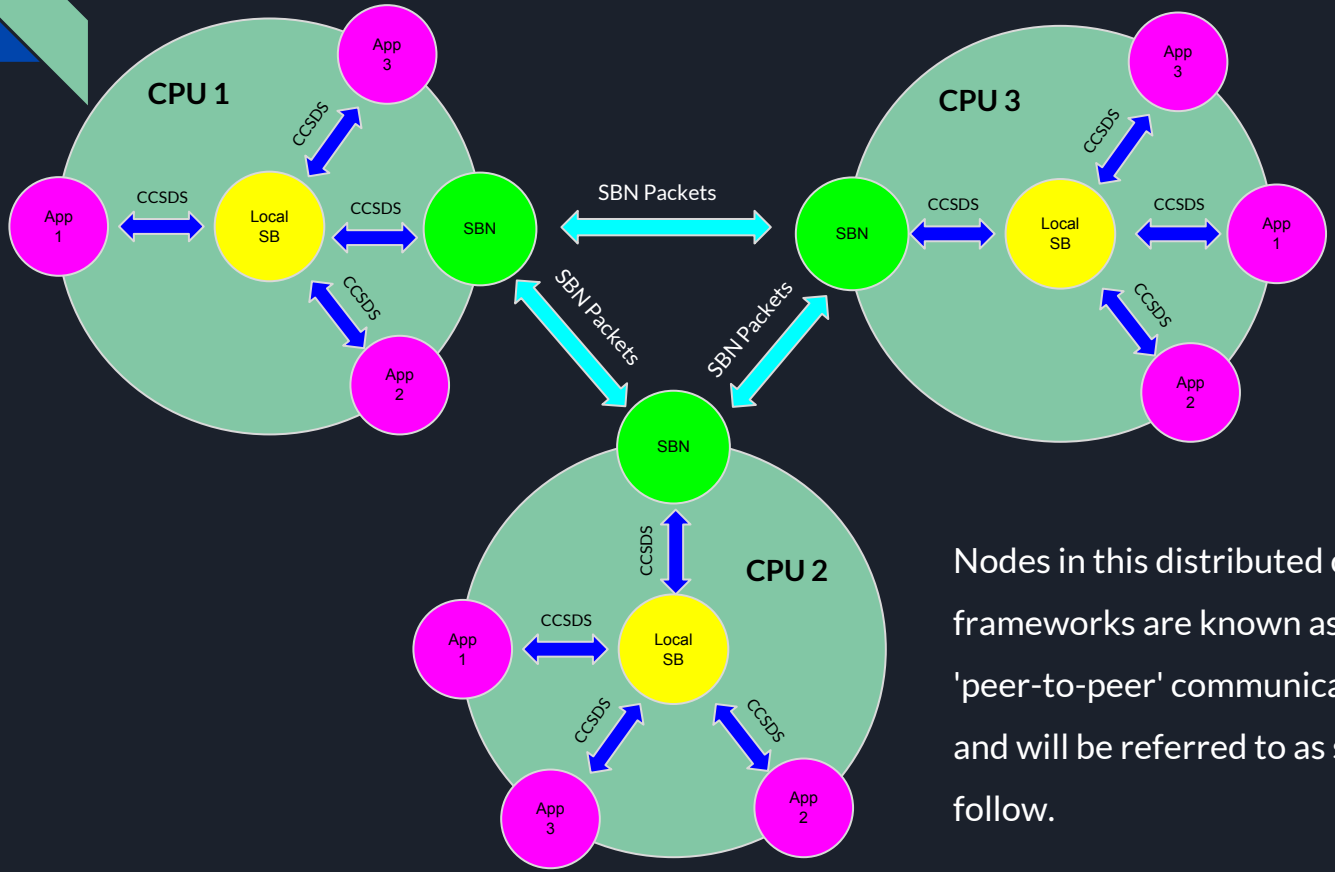
As this is designed to provide an extension to the local CFS Software Bus over a network interface, SBN-UDP is designed to work specifically with user-data encapsulated in CCSDS packets.

IV) The SBN-UDP Protocol



The SBN-UDP protocol ultimately consists of all nodes on the Software Bus Network sending SBN Packets to each other to maintain connection states, share CCSDS message ID subscriptions, and user-data.

IV) The SBN-UDP Protocol



Nodes in this distributed communication frameworks are known as 'peers' (this is a 'peer-to-peer' communication framework after all) and will be referred to as such in the notes that follow.

IV) The SBN-UDP Protocol

The SBN Packets may be 'heartbeats' (simply for maintaining connection state based on timeouts), 'announcements', 'subscriptions' and user data (CCSDS packets). Each packet has a 7-byte, big-endian header consisting of the following fields, respectively:

- 2-byte integer 'payload size'

- 1-byte integer 'message type'

- 4-byte integer 'CPU ID'

IV) The SBN-UDP Protocol

The 1-byte 'message type' field can be one of the following values:

- SBN_NO_MSG = 0x0
- SBN_SUB_MSG = 0x1
- SBN_APP_MSG = 0x2
- SBN_HEARTBEAT_MSG = 0xA0
- SBN_ANNOUNCE_MSG = 0xA1

IV) The SBN-UDP Protocol

For 'heartbeats' and 'announcement' messages, the SBN Packet does not need a payload. It can simply be a 7-byte header with no payload. Any payload attached to 'heartbeats' or 'announcements' will simply be ignored.

IV) The SBN-UDP Protocol

For 'peer subscription' messages, the payload is described as an 'SBN Subscription Packet.' Each subscription packet is big-endian with the following fields, respectively:

- 48-byte 'version' treated as a byte-array

- 2-byte 'subscription count'

- An array of subscriptions where each subscription is a 32-byte MID followed by a 2-byte QOS (currently QOS does not affect reliability of the networking protocol)

IV) The SBN-UDP Protocol

The 'peer connection' state is binary - peers are either 'connected' or 'disconnected'.

The peer connection state for an SBN node is determined by the last time that 'this' node received *any* SBN packet from a peer (could be a heartbeat, subscription, or user-data). A peer who hasn't send a message to another within the timeout will be considered to be disconnected by that peer. Similarly, a peer who considers another to be disconnected should immediately consider it connected upon receipt of *any* new SBN packet.

IV) The SBN-UDP Protocol

When a node starts up on SBN, it considers all of its peers to be in a 'disconnected' state. It will send an announcement message (or any message such as a heartbeat would be fine) to all peers. Upon the first receipt of a message from a peer, the node will consider that peer to be in a 'connected' state and send any of it's CCSDS message subscription information that peer.

As long as a peer is in a connected state, SBN will continue to send heartbeats periodically to maintain the connection state regardless of whether the user has data to send or receive. It's important that heartbeats are sent to connected peers at frequency greater than the connection timeout so the connection is not broken. For peers in a 'disconnected' state, a node will send less frequent announcements in case one or more of the heartbeat packets was dropped causing the connection to timeout. This makes the connection a bit more reliable.

IV) The SBN-UDP Protocol

When an application publishes a message from a particular node on SBN, the condition for sending that message to another node (or peer) on SBN is two-fold:

- That peer has subscribed to that message

- That peer is in a 'connected' state (we've received a message more recently than the connection timeout)

IV) The SBN-UDP Protocol

SBN makes no attempt to guarantee delivery of messages to connected peers over SBN-UDP or notify senders/receivers of failure. With this in mind, it is not recommended to use SBN over wireless networks (WIFI) where packet loss can be very significant.

V) Protobetter Serialization Library and Data Representation in CFS

Though CCSDS is the packet format used for inter-app communication throughout the CFS framework (in SB & SBN), the data representation for user-data portion of the CCSDS packet is entirely user-defined. Currently, SBN is designed to handle all serialization/deserialization of user-data across network interfaces using Protobetter - a serialization library designed for real-time flight software.

V) Protobetter Serialization Library and Data Representation in CFS

Protobetter works off of a language and platform neutral message definition schema called 'prototypes'. Here's an example of a message definition:

```
"{
        "typeName":"LilBity_c",

        "Members":
        [
                {"name":"a", "type":"float"},
                {"name":"b", "type":"uint32_t", "bits":31},
                {"name":"c", "type":"uint32_t", "bits":1},
                {"name":"d", "type":"uint32_t", "bits":4},
                {"name":"e", "type":"Vector_c", "arraylen":3},
                 {"name":"f", "type":"int16_t", "bits":16}
        ]
}"
```

V) Protobetter Serialization Library and Data Representation in CFS

Protobetter currently has two implementations - a c-implementation w/ a python compiler which generates serialization code in C from the prototype definitions as well as a c++ version which builds up a graph data structure during initialization & pre-computes bit/byte offsets for data so it can read/write directly to/from serialized data buffers.

V) Protobetter Serialization Library and Data Representation in CFS

The C-implementation's python compiler is currently integrated with CCDD via a python scripts w/ a custom code generation class for handling CCSDS packets containing protobetter-encoded user-data. The CCDD python script simply converts the CCDD message definitions to the python equivalent of a 'prototype' JSON object and runs both the default protobetter code generation class as well as the custom code generation class supplied specifically for integration in CFS/SBN.

For more information, check out the protobetter-c repo on Thales Gitlab server as well as the protobetter-dynamic repo on ESGL Gitlab server at JSC.