MORPH BRIDGE

Main idea: The idea for the bridge was to allow for channelled communication between different partners with ease. On a stroke of inspiration, the idea 'morphed' into morph-bridge. The idea of morph-bridge is to provide channelled communication between actors over any communication protocol. The idea will be open source so that any person who wants to set up a fast communication server for multi-protocol, multiplatform, cross-device, light-weight, customizable and free communication can just load the app on their vm and have the applications running on the go.

The main goals of the project would be to

* communicate securely across multiple platforms and devices
* communicate securely on multiple protocols.
* ensure communication is not only fast but secure
* have a fast, easy to use, bootstrappable api for fast communication

The main target protocols for integration are to be:

* zmq
* tcp
* websockets
* http get/post requests
* http SSE
* sms \*
* mqtt \*\*
* rabbitmq \*\*
* any other protocol that can be implemented.

\* will develop a skeleton which can be modified for different gateway apis. Only apis available on nodejs will be used for now.

\*\* need further research to determine applicability.

A summary of the app structure has been put on the draw.io file in the folder. The application will include a number of features that will achieve the goals stated above. These are :

* session management
* channel timeout feature (each channel will die after a timeout). This hopefully will increase security and improve memory management.
* Each node on the app will handle communication to/from a certain comm protocol, however, there is a global module that will be available for all nodes. This module will contain centralized methods for internode communication, channel creation and management, session management and data collection (usage stats not personal info)
* The nodes will not contain buffers where need not be. This will increase security of communication, however, if a protocol needs a buffer then a timed buffer will be used for each channel on that protocol.
* Channel access will be secured by an encrypted password access.. where ssl can be used, it will be used however this is not necessary as data is only passed to members of a channel.

To improve application performance and reliability, a mesh architecture will be used. Each node will be independent and will publish msgs received to all nodes subscribed to a channel. Therefore if any other node is added, it will automatically receive the data. There is no central node, however a separate node might be considered for node respawning or error management. However, for now this could be integrated into the global module and managed through it.

The functions of each channel will be written below. This is still a work in progress as most of it will be based on previous experience on the first bridge and chat apps and is subject to improvement.

Channel Class:

* create channel / if no created instance exists
* check if channel exists/ if not do nothing (in the instance a message is received)
* error handling to error log and to
* channel destruction after timeout (possibly set allowance for setting timeout.)
* check if channel subscribers exist, if not do nothing (in the instance a message is received)
* delete all subscribers when a channel is destroyed
* Check if subscriber exists, carry sender info to allow for return messaging. (possibly have external node managing this.)
* Send message to one subscriber of a channel. for app-client communication (error messaging, response handling … e.t.c.)
* set timeout.
* Check message size, check if size exceeds maximum for protocol/device parameters, if not send back msg to client to reduce size. e.g. in the instance of sms/low memory devices.
* Check password.

Communication Class:

* process message i.e. parse message/data into object that contains sender info. (ipc)
* publish message (ipc)
* dress error message and send to client. (will use error codes to reduce msg size.)
* Log handling, debug/error/status e.t.c. (will be used by janitor node)
* Subscribe channels to ipc sockets.

Session Management:

Is not necessary for now since sessions are already handled within the channel class i.e. each channel is timed.

Error Messages/Handling:

A node will be available to handle errors and sending errors. Each node should be able to send messages to a single client over a socket. They will have an ipc socket for errors, if there is an error the node will receive the msg, and send a message to a particular client over the related protocol. The errors generated can be: (each msg will be coded)

* Channel access not granted. (i.e. password wrong) /subscription wrong.
* Server Error.
* Message size exceeded.
* Subscription successful/ Channel created

# General Notes

Care must be taken to ensure non duplication of channels and channel subscribers. Instantiation of channels must be full proof across all nodes i.e. if a channel is created in one node, it is created to all nodes. ensuring any other device addition will be able to access the same info.

Password must be handled such that no unencrypted data is saved to a channel. The password should be used when a subscriber tries to send a message to a protocol. The pass will be checked to ensure it has access to a channel, if yes publish message, if not reply to user, he has no access.