Location Mechanisms

- Main issue: How does the AWS instance locate the device? Because the AWS instance can have a public IP, the device can locate the AWS instance. As the device **does not** have a public IP, it is harder for AWS to locate the device.
- 3 options (4 when including Rajat's suggestion):
  - Use a VPN layer
  - Change the style of communication between the device and the AWS instance
  - Use websockets to initialize a connection, and lock onto that connection for future connections.
  - Rajat's suggestion involving websockets

Using a VPN layer - Option #1

- The purpose of the VPN layer is to bring the AWS instance and the device into a shared, secure local network.
- Relies on an AWS instance to act as the VPN server. This can be either the main AWS instance running the nav goals programs, or another separate AWS instance.
  - Possible to acquire an OpenVPN Access Server AWS instance for free
  - So far, the first option has been tested already (running the server in the same instance as the main programs). The latter option is still being tested.
- Pros:
  - Secure connection, does not force us to make the main AWS instance have a public IP
  - If multiple devices are on the same local network, even if the the main AWS instance fails, as long as the server is up and running, the devices can communicate with each other.
- Cons:
  - Using a VPN on the device adds overhead!

Changing the communication style - Option #2

- Current style:
  - The AWS instance is the client, and each ROS-running device is a server.
  - The AWS instance will run continuous client-server calls in the background, and then will pull out the latest stored information on the respective ROS devices when desired.
  - It requires the AWS instance to be able to locate the devices on its own.
- Proposed change:
  - The AWS instance will be the server, and the ROS-running devices will each be a client.
  - The devices will continuously send information into the AWS instance as client requests, and receive nav goals as the completion of the client-server call.
  - Requires the AWS instance to have a constant public IP.

Using websockets to initialize device IPs - Option #3

- In theory:
  - At the start of the run, each ROS-running device will bind to a port on the AWS instance using websockets. The AWS instance will store the (port number/IP address, ROS device name) key-value pairs. The instance will continue to act like a client, and will send requests based on the stored IP addresses.
- Potential problem in implementation - not quite sure if it will work out how I've theorized.
- Maybe Rajat has an idea / experience with implementing this, or a similar solution?