PainlessMesh

Intro to painlessMesh

painlessMesh is a library that takes care of the particulars of creating a simple mesh network using esp8266 and esp32 hardware. The goal is to allow the programmer to work with a mesh network without having to worry about how the network is structured or managed.

True ad-hoc networking

painlessMesh is a true ad-hoc network, meaning that no-planning, central controller, or router is required. Any system of 1 or more nodes will self-organize into fully functional mesh. The maximum size of the mesh is limited (we think) by the amount of memory in the heap that can be allocated to the sub-connections buffer and so should be really quite high.

JSON based

painlessMesh uses JSON objects for all its messaging. There are a couple of reasons for this. First, it makes the code and the messages human readable and painless to understand and second, it makes it painless to integrate painlessMesh with javascript front-ends, web applications, and other apps. Some performance is lost, but I haven't been running into performance issues yet.
Converting to binary messaging would be fairly straight forward if someone wants to contribute.

Wifi & Networking

painlessMesh is designed to be used with Arduino, but it does not use the Arduino WiFi libraries, as we were running into performance issues (primarily latency) with them. Rather the networking is all done using the native esp32 and esp8266 SDK libraries, which are available through the Arduino IDE. Hopefully though, which networking libraries are used won't matter to most users much as you can just include painlessMesh.h, run the init() and then work the library through the API.

painlessMesh is not IP networking

painlessMesh does not create a TCP/IP network of nodes. Rather each of the nodes is uniquely identified by its 32bit chipId which is retrieved from the esp8266/esp32 using the `system_get_chip_id()` call in the SDK. Every node will have a unique number. Messages can either be broadcast to all of the nodes on the mesh, or sent specifically to an individual node which is identified by its `nodeId.

Limitations and caveats

- Try to avoid using `delay()` in your code. To maintain the mesh we need to perform some tasks in the background. Using `delay()` will stop these tasks from happening and can cause the mesh to lose stability/fall apart. Instead we recommend using TaskScheduler which is used in `painlessMesh` itself. Documentation can be found here. For other examples on how to use the scheduler see the example folder.
- `painlessMesh` subscribes to WiFi events. Please be aware that as a result `painlessMesh` can be incompatible with user programs/other libraries that try to bind to the same events.
- Try to be conservative in the number of messages (and especially broadcast messages) you sent per minute. This is to prevent the hardware from overloading. Both esp8266 and esp32 are limited in processing power/memory, making it easy to overload the mesh and destabilize it. And while `painlessMesh` tries to prevent this from happening, it is not always possible to do so.
- Messages can go missing or be dropped due to high traffic and you can not rely on all messages to be delivered. One suggestion to work around is to resend messages every so often. Even if some go missing, most should go through. Another option is to have your nodes send replies when they receive a message. The sending nodes can the resend the message if they haven’t gotten a reply in a certain amount of time.

Installation

`painlessMesh` is included in both the Arduino Library Manager and the platformio library registry and can easily be installed via either of those methods.

Dependencies

painlessMesh makes use of the following libraries, which can be installed through the Arduino Library Manager

- ArduinoJson
- TaskScheduler
- ESPAsyncTCP (ESP8266)

- AsyncTCP (ESP32)

If platformio is used to install the library, then the dependencies will be installed automatically.

Examples

StartHere is a basic how to use example. It blinks built-in LED (in ESP-12) as many times as nodes are connected to the mesh. Further examples are under the examples directory and shown on the platformio page.

Development on your own machine

After cloning the repository, you will need to initialize and update the submodules.

```
git submodule init
git submodule update
```

After that you can compile the library using the following commands

```
cmake -G Ninja
ninja
```

This will compile a number of test files under `./bin/catch_` that can be run. For example using:

```
run-parts --regex catch_ bin/
```

Getting help

There is help available from a variety of sources:

- The included examples
- The API documentation
- The wiki
- On our new forum/mailinglist
- On the gitter channel

Contributing

We try to follow the git flow development model. Which means that we have a `develop` branch and `master` branch. All development is done under feature branches, which are (when finished) merged into the development branch. When a new version is released we merge the `develop` branch into the `master` branch. For more details see the CONTRIBUTING file.

painlessMesh API

Using painlessMesh is painless!

First include the library and create an painlessMesh object like this.

```
#include <painlessMesh.h>
painlessMesh  mesh;
```

The main member functions are included below. **Full documentation can be found here**

Member Functions

void painlessMesh::init(String ssid, String password, uint16_t port = 5555, WiFiMode_t connectMode = WIFI_AP_STA, _auth_mode authmode = AUTH_WPA2_PSK, uint8_t channel = 1, phy_mode_t phymode = PHY_MODE_11G, uint8_t maxtpw = 82, uint8_t hidden = 0, uint8_t maxconn = 4)

Add this to your setup() function. Initialize the mesh network. This routine does the following things.

- Starts a wifi network
- Begins searching for other wifi networks that are part of the mesh
- Logs on to the best mesh network node it finds... if it doesn't find anything, it starts a new search in 5 seconds.

`ssid` = the name of your mesh. All nodes share same AP ssid. They are distinguished by BSSID. `password` = wifi password to your mesh. `port` = the TCP port that you want the mesh server to run on. Defaults to 5555 if not specified. `connectMode` = switch between WIFI_AP, WIFI_STA and WIFI_AP_STA (default) mode

void painlessMesh::stop()

Stop the node. This will cause the node to disconnect from all other nodes and stop/sending messages.

void painlessMesh::update(void)

Add this to your loop() function This routine runs various maintenance tasks... Not super interesting, but things don't work without it.

void painlessMesh::onReceive(&receivedCallback)

Set a callback routine for any messages that are addressed to this node. Callback routine has the following structure.

```
void receivedCallback( uint32_t from, String &amp;amp;msg )
```

Every time this node receives a message, this callback routine will the called. “from” is the id of the original sender of the message, and “msg” is a string that contains the message. The message can be anything. A JSON, some other text string, or binary data.

void painlessMesh::onNewConnection(&newConnectionCallback)

This fires every time the local node makes a new connection. The callback has the following structure.

```
void newConnectionCallback( uint32_t nodeId )
```

`nodeId` is new connected node ID in the mesh.

void painlessMesh::onChangedConnections(&changedConnectionsCallback)

This fires every time there is a change in mesh topology. Callback has the following structure.

```
void onChangedConnections()
```

There are no parameters passed. This is a signal only.

bool painlessMesh::isConnected(nodeId)

Returns if a given node is currently connected to the mesh.

`nodeId` is node ID that the request refers to.

void painlessMesh::onNodeTimeAdjusted(&nodeTimeAdjustedCallback)

This fires every time local time is adjusted to synchronize it with mesh time. Callback has the following structure.

`void onNodeTimeAdjusted(int32_t offset)`

`offset` is the adjustment delta that has been calculated and applied to local clock.

void onNodeDelayReceived(nodeDelayCallback_t onDelayReceived)

This fires when a time delay measurement response is received, after a request was sent. Callback has the following structure.

`void onNodeDelayReceived(uint32_t nodeId, int32_t delay)`

`nodeId` The node that originated response.

`delay` One way network trip delay in microseconds.

bool painlessMesh::sendBroadcast(String &msg, bool includeSelf = false)

Sends msg to every node on the entire mesh network. By default the current node is excluded from receiving the message (`includeSelf = false`). `includeSelf = true` overrides this behavior, causing the `receivedCallback` to be called when sending a broadcast message.

returns true if everything works, false if not. Prints an error message to Serial.print, if there is a failure.

bool painlessMesh::sendSingle(uint32_t dest, String &msg)

Sends msg to the node with Id == dest.

returns true if everything works, false if not. Prints an error message to Serial.print, if there is a failure.

String painlessMesh::subConnectionJson()

Returns mesh topology in JSON format.

std::list<uint32_t> painlessMesh::getNodeList()

Get a list of all known nodes. This includes nodes that are both directly and indirectly connected to the current node.

uint32_t painlessMesh::getNodeId(void)

Return the chipId of the node that we are running on.

uint32_t painlessMesh::getNodeTime(void)

Returns the mesh timebase microsecond counter. Rolls over 71 minutes from startup of the first node.

Nodes try to keep a common time base synchronizing to each other using an SNTP based protocol

bool painlessMesh::startDelayMeas(uint32_t nodeId)

Sends a node a packet to measure network trip delay to that node. Returns true if nodeId is connected to the mesh, false otherwise. After calling this function, user program have to wait to the response in the form of a callback specified by `void painlessMesh::onNodeDelayReceived(nodeDelayCallback_t onDelayReceived)`.

nodeDelayCallback_t is a function in the form of `void (uint32_t nodeId, int32_t delay)`.

void painlessMesh::stationManual(String ssid, String password, uint16_t port, uint8_t *remote_ip)

Connects the node to an AP outside the mesh. When specifying a `remote_ip` and `port`, the node opens a TCP connection after establishing the WiFi connection.

Note: The mesh must be on the same WiFi channel as the AP.

void painlessMesh::setDebugMsgTypes(uint16_t types)

Change the internal log level. List of types defined in Logger.hpp: ERROR | MESH_STATUS | CONNECTION | SYNC | COMMUNICATION | GENERAL | MSG_TYPES | REMOTE

Revision #1
Created 3 November 2024 12:13:18 by Giovan Christoffel Sihombing
Updated 3 November 2024 12:13:32 by Giovan Christoffel Sihombing