(prima diapositiva protocollo cominicazione) Introduzione

- Tipi di messaggio:
  1. DiscoveryReq. Inviata dal server ai suoi vicini e propagata in broadcast fra i nodi per costruire l'albero di copertura della rete ed evitare così la formazione di cicli

     `{ "from" : uint32_t idServer, "updat_number" : int currentUpdateNumber, "sender_id" : uint32_t prevHopId, "type" :  0 }`
  2. Data. Creati dai nodi con sensori vengono propagati o attraverso il nextHop noto oppure in broadcast nel caso in cui o il nextHop non è stato ancora trovato o la rotta è scaduta.

     `{ "from" : uint32_t idSourceStation, "id" : int currentPackageNumber, "temp" : float tempData, "type" :  1 }`

Intro pt.2

- Strutture dati e variabili fonamentali
  - `map<uint32_t ,int> lastSentMsg`. Dizionario attraverso il quale si evita la diffusione di pacchetti duplicati all'interno della rete. Il generico elemento è della forma:

    `< uint32_t stationId, int lastPackegeRecievedId >`
  - `update e lastSyncTime. G`estite dal server e usate nella fase di discovery per comunicare ai nodi l'aggiornamento delle rotte e l'avvio di una nuova fase di sincronizzazione.

Nodi

Sono tutti ESP ma, a seconda del loro ruolo all'interno della rete, si distinguono in:

- **station&LM35(GPIO2).** Il compito di nodi di questo tipo è quello di leggere i dati dai sensori, creare un pacchetto di tipo DATA e inviarlo al server.
- **genericNode.** I nodi intermedi si occupano esclusivamente di inoltrare i pacchetti dati verso il server e le discoveryReq a tutti vicini.
- **espServer.** Il/i server lanciano periodicamente una discoveryReq e si preoccupano di inviare sulla porta seriale tutti i ***nuovi*** pacchetti dati.

Discovery nel server

- Implementata da ***discoveryTree()***, crea e invia un pacchetto per reinizzializzare l'***albero di copertura*** della rete e risincronizzare i nodi

```
void discoveryTree(){
  char msg[256];
  sprintf(msg, "{\"from\": %d, \"update_number\": %d, \"sender_id\": %d,
  \"type\": 0}", mesh.getChipId(), ++update, mesh.getChipId());
  /*Prevent overflow*/
  if(update == INT_MAX)
    update = 0;
  String p(msg);
  mesh.sendBroadcast(p);
  lastSyncTime = mesh.getNodeTime();
  return;
}
```

Discovery nei nodi

- Se la ***discoveryReq*** ricevuta è più recente dell'ultima memorizzata aggiorno la rotta e propago la richiesta in broadcast

```
void receivedCallback( uint32_t from, String &msg_str ){
  JsonObject& msg = jsonBuffer.parseObject(msg_str);
  int type = msg["type"];
  switch(type){
    case(DISCOVERY_REQ):{
        if(msg["update_number"] > update){
          update = msg["update_number"];
          if(update == INT_MAX)
            /*Prevent overflow*/
            update = 0;
          nextHopId = msg["sender_id"];
          propagateDiscovery(msg);
         lastSyncTime = mesh.getNodeTime();
        }
    }break;
...
```

Gestione pacchetto dati nel server

- Se il pacchetto ricevuto da **`from`** è più recente dell'ultimo memorizzato, lo invio sulla seriale e aggiorno il dizionario.

```
void receivedCallback( uint32_t from, String &msg_str ){
  JsonObject& message = jsonBuffer.parseObject(msg_str);
  int type = message["type"];
  if(type!=DISCOVERY_REQ){
    if (lastSentMsg[message["from"]] != NULL && lastSentMsg[message["from"]] != 0)
      if(lastSentMsg[message["from"]] < message["id"]){
        addAlreadySent(message["from"], message["id"]);
        printJson(message);
      }
    else{
        addAlreadySent(message["from"], message["id"]);
        printJson(message);
    }}}
```

Gestione pacchetto dati nei nodi

- Stessa storia del server solo che invece che inviarlo sullas seriale lo invio o al ***nextHop***, se esiste ed è valido, oppure in ***broadcast***.

```
void receivedCallback( uint32_t from, String &msg_str ){
  JsonObject& msg = jsonBuffer.parseObject(msg_str);
  int type = msg["type"];
  switch(type){
   …
    case(DATA):{
      if(lastSentMsg[msg["from"]] != NULL &&
          lastSentMsg[msg["from"]] != 0)
        if(lastSentMsg[msg["from"]] < msg["id"])
          propagateData(msg_str, msg["from"], msg["id"]);
      else
        propagateData(msg_str, msg["from"], msg["id"]);
    }break;
    default:{}break;
  }
}
```

```
void propagateData(String& msg_str, uint32_t from, int id ){
  /*If the route is expired, nextHopId is set to -1*/
  if((mesh.getNodeTime()-lastSyncTime)>=SYNCINTERVAL)
    nextHopId = -1;
  if(nextHopId != -1)
    mesh.sendSingle(nextHopId, msg_str);
  else
    mesh.sendBroadcast(msg_str);
  if(id==99)
    /*Prevent overflow*/
    lastSentMsg[from] = -1;
  else
    lastSentMsg[from] = id;
  return;
}
```