



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Mirjam Trippel

s0572665

Drahtlose Netzwerke

Konnektivität mit Telegram

Kurzbeschreibung:

Das Projekt nutzt einen ESP8266 und einen DHT22 Sensor. Damit werden Temperatur und Feuchtigkeit Messwerte abgetastet.

Diese Messwerte werden dem Raspberry Pi über ein Netzwerk mit MQTT übermittelt.

Der Raspberry Pi ist ein Gateway zwischen MQTT und Telegram.

Mit Node-red ist es möglich, in Telegram diese Messwerte abzufragen.

Es werden auch die Telegram-Nodes erklärt und vorgestellt.

Inhaltsverzeichnis:

- Ist- und Soll-Zustand
- Skizzen & Fotos vom ESP8266 + DHT22 Aufbau
- Einführung
- Ausblick
- Verwendete und Benötigte Node-Red-Nodes
- Link zum Repo
- Quellenverzeichnis
- Codebeispiel

Ist-Zustand:

Der physische Aufbau besteht aus 2 Komponenten. Zuerst wird ein Breadboard mit Verbindungskabeln benötigt. Welche den ESP8266 und den DHT22 verbinden. Das Datenübertragungskabel braucht ein 10k Ohm Widerstand zur positiven Stromzufuhr, um falschen Messwerten vorzubeugen. Der ESP8266 wird vom Raspberry mit Strom durch ein USB zu MicroUSB Kabel versorgt. Der Raspberry muss nur noch an einen Monitor, Netzteil, Tastatur und eine Maus angeschlossen werden.

Danach wird nur noch ein Netzwerk benötigt, wo sich der ESP und der Raspberry einwählen können. Der Raspberry benötigt für Telegram eine Verbindung zum Internet.

Der Software Aufbau ist sehr simpel. Der ESP8266 ist mit der Micropython Stable Firmware - 1024kb geflashed. Die Python MQTT Implementation wurde von Dr. Alexander Huhn zur Verfügung gestellt. Vielen lieben Dank. Beim ESP8266 müssen nur das Netzwerk mit dem Secrets Login, die Netzwerk IP vom Raspberry und die MQTT Topics, auf die der Raspberry hört, hinterlegt werden.

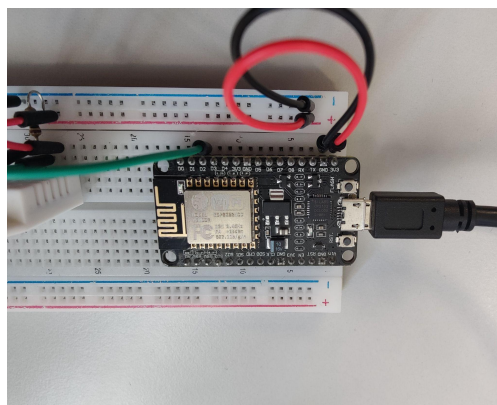
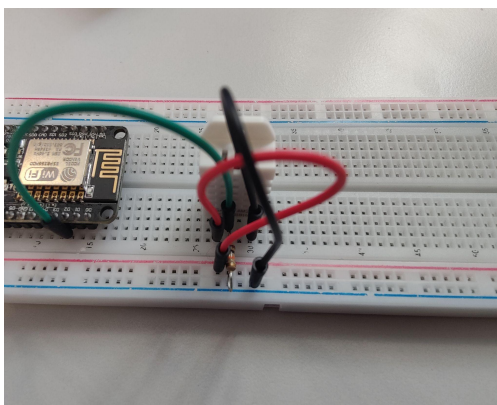
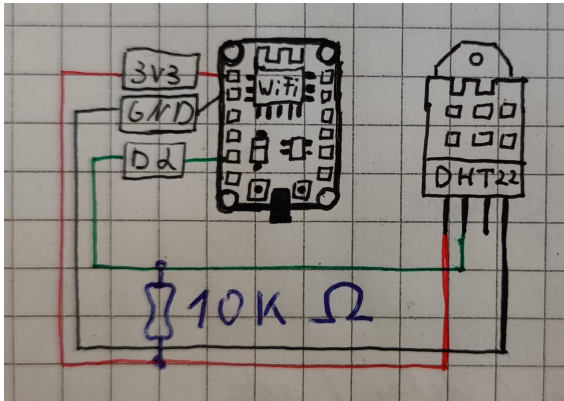
Der Raspberry braucht nur MQTT und Node-red mit den notwendigen Nodes zu Installieren. Die verwendeten Nodes werden am Ende des Dokuments Aufgelistet.

Soll-Zustand:

Der ist erreicht und man kann nach eigenem Ermessen mehr ESP8266+Sensoren und weitere Node-red MQTT/Telegram Funktionen oder Datenbanken hinzufügen.

Das Beispiel, in dem auf die Sensor Messwerte mit einem Telegram InlineKeyboard zugegriffen wird, wurde auch erfolgreich implementiert.

Skizzen & Fotos vom ESP8266 + DHT22 Aufbau:

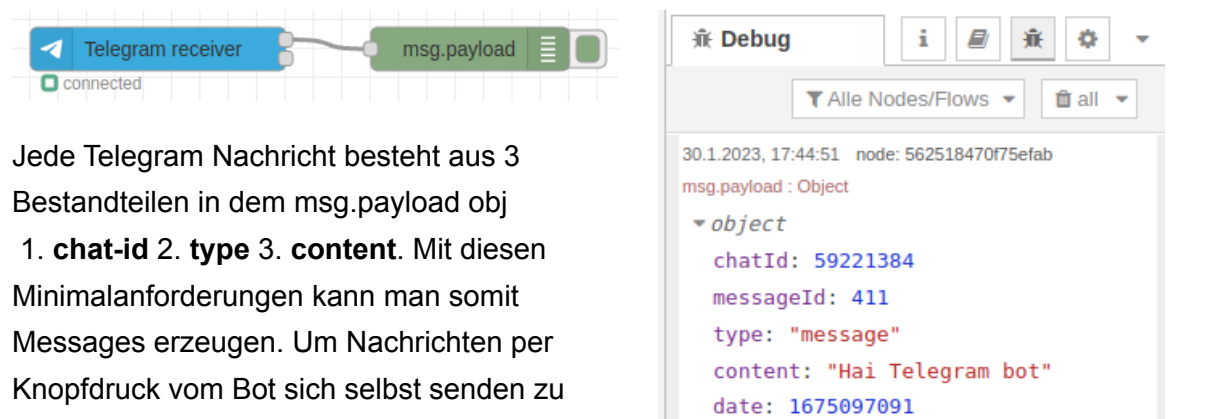


Einführung:

Nachdem MQTT funktioniert und der Raspberrypi die Sensor Messwerte richtig empfängt, kann man sich mit den Telegram-Nodes auseinandersetzen. Jetzt sollten alle Node Module installiert sein. Die Liste der Module ist am Ende des Dokuments zu finden. Es gibt auch die Node-red-telegram-dokumentation[1], diese habe ich auch hauptsächlich verwendet und mit vielen selbst Experimenten, Node-red erkundet.

Bevor es losgehen kann, braucht man als allererstes einen Telegram-Bot, der von den Telegram-Nodes angesprochen wird. Den bekommt man durch den “@BotFather” in Telegram. Da wählt man einen Namen, Benutzernamen und bekommt danach einen HTTP-API Token. Mit dem Token kann man sich Authentifizieren und dies ist, neben dem Namen vom Bot, das einzig Wichtige was man fürs erste braucht um die Sending und Receiving Nodes einzurichten. Wenn man in Node-red ein Profil mit den Parametern vom BotFather erstellt hat, loggen sich die Nodes dann ein und greifen auf den erstellten Bot zu.

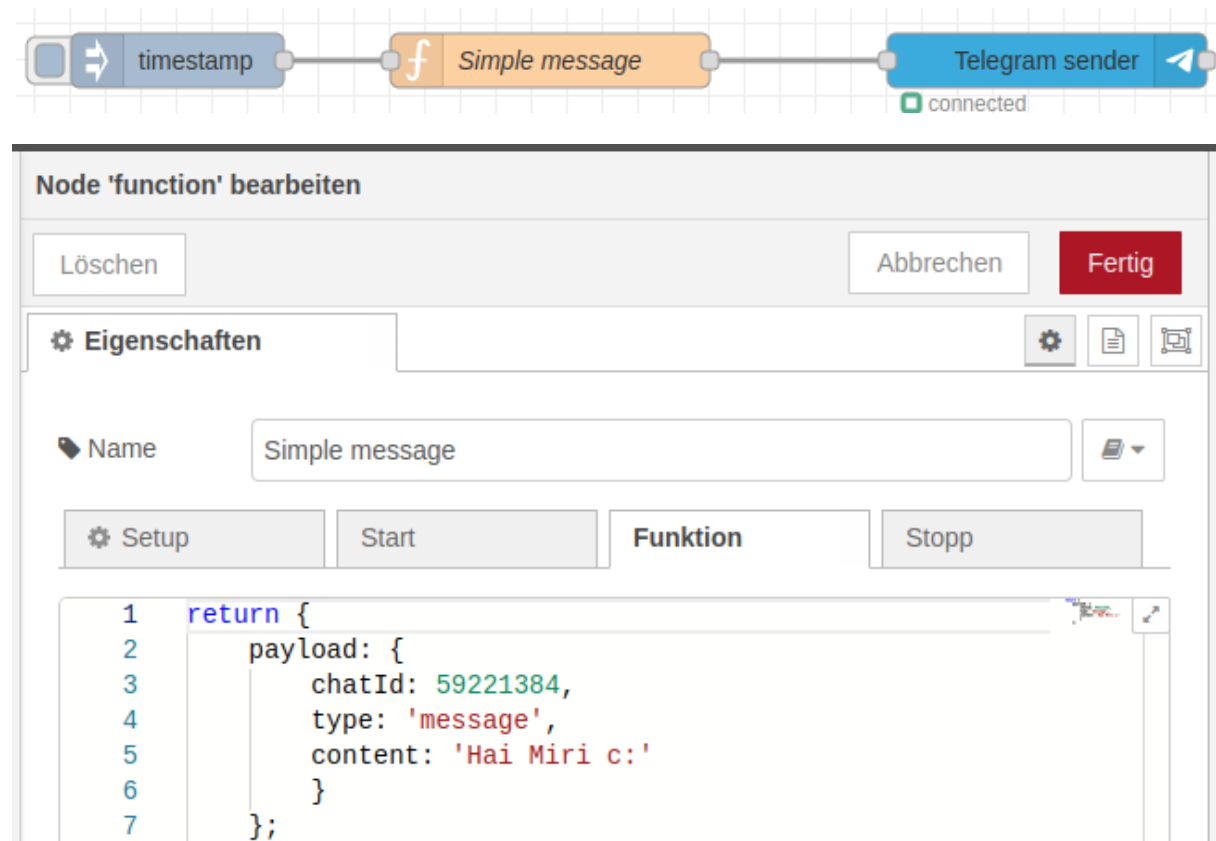
Er läuft quasi immer, wenn Node-red angemacht wird. Die Receiver und Sender Nodes verweisen auch während des Einrichtens auf den BotFather, für weitere Informationen. Nachdem einrichten des Bot Profils für die Nodes, kann man schon mit einer debugNode, sich die ankommenden Nachrichten in der Node-red Konsole ausgeben lassen und bekommt so die Persönliche chat-id heraus. Indem man den frisch erstellten Bot in Telegram anschreibt. Das Erste Problem ist nämlich, um direkte Nachrichten einem User in Telegram schicken zu können, braucht man eine "Konversation". Diese Konversation kann vom User initialisiert werden und genau das ist die ChatID. Die ChatID wird dann als Parameter für direkte Nachrichten immer benötigt und ist automatisch in den Nachrichten vom User zum Bot in dem msg.payload OBJ enthalten. So bekommt man sie dann auch:



Jede Telegram Nachricht besteht aus 3 Bestandteilen in dem msg.payload obj

1. **chat-id**
2. **type**
3. **content**.

Mit diesen Minimalanforderungen kann man somit Messages erzeugen. Um Nachrichten per Knopfdruck vom Bot sich selbst senden zu können, kann man beispielsweise die Inject-node verknüpfen, mit der selbstgemachten Direkten Nachricht in der Function-Node. Und dann sendet Node-red problemlos Nachrichten direkt aufs Handy.



Node 'function' bearbeiten

Löschen Abbrechen Fertig

Eigenschaften

Name Simple message

Setup Start Funktion Stopp

```

1 return {
2   payload: {
3     chatId: 59221384,
4     type: 'message',
5     content: 'Hai Miri c:'
6   }
7 };

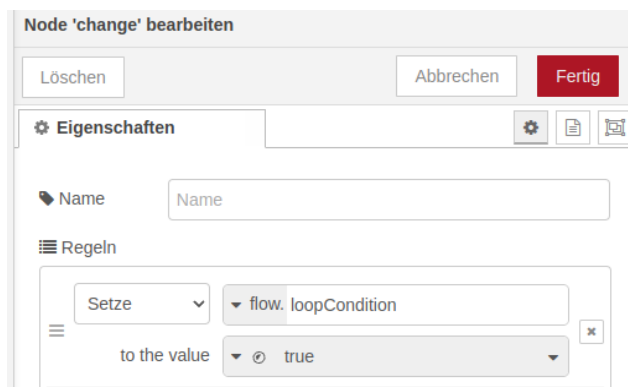
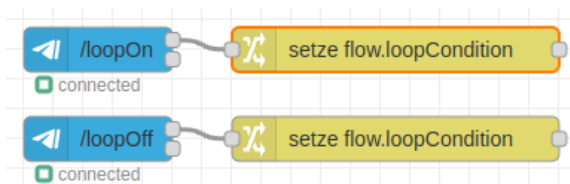
```

Die simpelste Telegram-node Konstellation ist eine Feedback Loop, indem man die Receiver- und Sender-nodes miteinander verbindet.

Ich habe versucht, einen State durch Telegram zu triggern und bin dabei auf ein zweites Problem gestoßen. In Node-red gibt es keine Möglichkeit etwas zu pausieren und dann später weiter fortzuführen, es gibt keinen "Pause" Knopf. Es gibt aber die Möglichkeit mit einer "If-anweisung" sowas ähnliches zu erzeugen. Eine Art Baton/Flag, die je nach State gewisse Abläufe weiter funktionieren lässt.

Also wurde die Idee des Feedback Loops mit dem State Triggern gemischt und umgesetzt mit Flow/Global Context und den Command-nodes.

Um einen State im Flow/Global Context zu setzen, gibt es die Change-node.



Da gibt es nur ein Problem. Man kommt nicht einfach an den Gesetzten State wieder heran. Da habe ich mich mehr mit Context beschäftigt[2] und herausgefunden, dass man mit `flow.get("loopCondition")`

darauf zugreifen kann. Die Command-Nodes haben auch die Möglichkeit, registriert zu werden. Das bedeutet, dass es ein Dropdown-Menü neben der Chat-Funktion gibt. Dann können Commands mit einem Klick aufgerufen werden.

Flow Context ist ein Scope, in dem alle Nodes desselben Flows zugreifen können.

Global Context reicht über alle Flows hinaus, wurde aber nicht für diese Anwendung benötigt.

Allgemein ist es eine gute Sache, ein /help Command zu erstellen. Dieser dann eine Liste von verfügbaren Commands mit einer kurzen Beschreibung dem Nutzer zuschickt. Damit kann der Nutzer eine beschreibung aller Unterstützten Commands lesen und direkt auf sie Klicken zum ausführen. Registrierte Commands machen aber fast dasselbe, nur haben sie nicht so viel Platz für längere/ausführlichere Beschreibungen. Die Message kann mit String Konkatination gemacht werden und mit /help könnt ihr das euch gerne ansehen.

Die Event-Nodes sind zuständig, um auf Aktionen vom Chat zu interagieren.

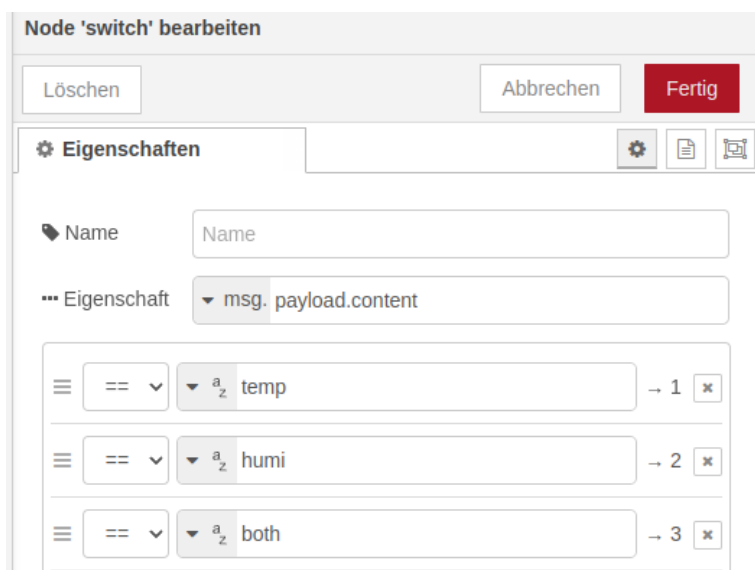
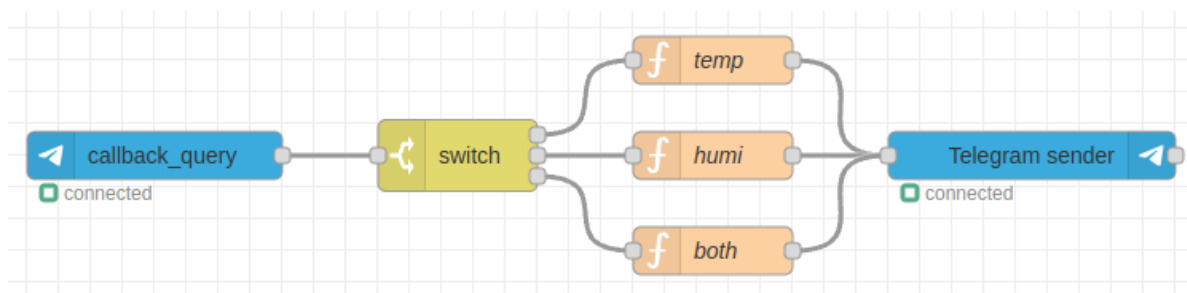
Beispielsweise wenn jemand eine Nachricht bearbeitet, die schon geschrieben wurde. Sei es ein Text von einem Bild, der Text einer Nachricht oder der Text von einer Antwort auf eine vorherige Nachricht. Die Event-Node kann auch auf besondere Nachrichten Reagieren, wie beispielsweise eine Umfrage oder ein InlineKeyboard.

Dafür wurde sie auch verwendet, um die User Action eines Buttons vom InlineKeyboard mit einem Switch Case an den entsprechenden access und return command weiter zu leiten.

Das Code Beispiel vom InlineKeyboard[A] ist nach dem Quellenverzeichnis zu finden.

Ein InlineKeyboard ist ein Objekt was einer Nachricht angehängt wird. Es sind die “Knöpfe”, die unter der Nachricht auftreten[3]. Diese Knöpfe können zum Beispiel eine Web App öffnen, einen Link aufrufen, man kann sich auch darüber Authentifizieren und man kann Zahlungen ausführen, alles per Knopfdruck. Eine Authentifizierung kann beispielsweise mit dem Telegram @UserName funktionieren oder durch den Knopf eine HTML Website/Formular verwenden.

Für das Projekt habe ich ein simples InlineKeyboard erzeugt welches mit einem Command aufgerufen wird und dessen verwendung, durch die Callback-query-node abgefangen wird und an die richtigen Function-nodes weiter leitet.



Die Temp, Humi, Both Function Nodes, nutzen die Flow-Variablen und erstellen mit ihrem Inhalt eine direkte Nachricht an den User.

Ausblick:

Man könnte vieles an diesem Projekt weitermachen. beispielsweise die Sicherheit mit MQTT security verbessern oder eine Authentifizierungen seitens Telegrams, um nur gewissen Nutzern vollen funktionsumfang zu gewährleisten.

Man könnte auch Messwerte über eine gewisse Zeit in einem Array/einer Datenbank speichern und diese statistisch dann auswerten.

Eine schöne Idee wäre, dass man eine Tür mit einem ESP und Servomotor über Telegram öffnen und schließen würde.

Verwendete und Benötigte Node-Red-Nodes:

Node-red 3.0.2
Node-red-contrib-string 1.0.0
Node-red-contrib-telegrambot 14.8.7
Node-red-dashboard 3.2.3

Link zum Repo:

<https://github.com/MirjamTrippel/DrahtloseNetzwerke>

Quellenverzeichnis:

- [1] - <https://flows.nodered.org/node/node-red-contrib-telegrambot>
- [2] - <https://stevesnoderedguide.com/node-red-variables>
- [3] - <https://core.telegram.org/bots/features#inline-keyboards>
- [4] -

Code beispiel:

[A] - InlineKeyBoard Code für eine Function-Node, die in eine Sender-Node zeigt:

```
var keyboard = {
  reply_markup: JSON.stringify({
    "inline_keyboard": [
      [
        {
          "text": "Temperature",
          "callback_data": "temp"
        },
        {
          "text": "Humidity",
          "callback_data": "humi"
        },
        {
          "text": "All data",
          "callback_data": "both"
        }
      ]
    ]
  })
};

msg.payload.content = "This keyboard accesses the sensory data:";
msg.payload.options = keyboard;
msg.payload.type = "message";
return [msg]
```