Internet of Things – Datenaustausch zwischen Messstationen

Gesucht wird eine Möglichkeit des Datenaustausches trotz evtl nicht hoch performanter Verbindungen (im Gegensatz zum Echtzeitverfahren bei zB Feldbussen)

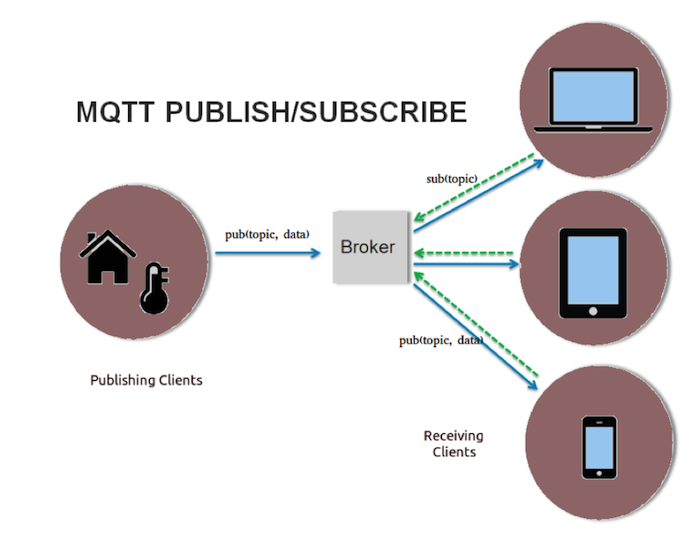
Beim IoT sind unterschiedlichste Partner verbunden zB Sensoren, Aktoren, Mikrocontrollersysteme, PCs, embedded Systems zB in Autos, Handys,…  
Sensoren publishen (veröffentlichen/senden) Daten, die sie gemessen haben

Aktoren subscriben (interessieren sich für) Steuerbefehle, um ein- oder ausgeschaltet zu werden

Eine Lösung ist das MQTT-Protokoll (port 1883 bzw 8883 secure mit TLS) MQTT message queuing telemetry transport

Zentraler Umschaltplatz der ausgetauschten Informationen ist der Broker (=Server)

Clients senden dem Server (“Broker”) nach Verbindungsaufbau Nachrichten mit einem Topic, das die Nachricht mit / abgestuft hierarchisch spezifiziert zB /Villach/HTL/EDV6/Temperatur. Dazu kommt der eigentliche Nachrichteninhalt (payload), in diesem Fall also der Zahlenwert und die Einheit der Temperatur also zB { “payload“:“21.2 Grad Celsius“ } am besten als json string man kann aber auch generische Zeichenketten basteln. Das Senden nennt man publishen.



Andere Clients können diese Topics abonnieren, wobei der Server die empfangenen Nachrichten an die entsprechenden Abonnenten weiterleitet.

Man kann mit der wildcard # auch gleich mehrere Topics abonnieren, zB alle Topics die mit htlvillach beginnen mit /htlvillach/#

Nachrichten werden mit einem definierbaren Quality of Service versendet:

* Stufe 0: at most once (die Nachricht wird einmal gesendet und kommt bei Verbindungsunterbrechung möglicherweise nicht an)
* Stufe 1: at least once (die Nachricht wird so lange gesendet, bis der Empfang bestätigt wird, und kann beim Empfänger mehrfach ankommen)
* Stufe 2: exactly once (hierbei wird sichergestellt, dass die Nachricht auch bei Verbindungsunterbrechung genau einmal ankommt).

Außerdem kann mit dem Retain-Flag der Server angewiesen werden, die Nachricht zu diesem Topic zwischenzuspeichern. Clients, die neu auf dieses Thema abonnieren, bekommen als erstes die zwischengespeicherte Nachricht zugestellt.

Beim Verbindungsaufbau können Clients einen „letzten Willen“ in Form einer Nachricht definieren. Falls die Verbindung zum Client verloren geht, wird diese Nachricht publiziert und dabei an die entsprechenden Abonnenten gesendet.

MQTT wird üblicherweise über TCP benutzt und hat einen 2-Byte-Header. Das erste Byte enthält den Nachrichtentyp (4 Bit), den Quality of Service (2 Bit) und ein Retain-Flag.

Die wichtigsten Nachrichten-Typen sind:

* CONNECT
* CONNACK
* PUBLISH
* SUBSCRIBE
* UNSUBSCRIBE
* DISCONNECT

Das zweite Byte enthält die Länge des restlichen MQTT-Pakets.

Daran schließt sich ein variabler Teil an, der das MQTT-Topic, also das Thema enthält. Abschließend kommt der Payload, also der Dateninhalt, der unter dem Thema veröffentlicht wird.  
Der Payload kann auch komplizierter sein zB eine Videosequenz oder Tonaufzeichnungen usw

Mit den Zeichen # kann ab einer Hierarchie-Ebene alles, und was darunter liegt, empfangen werden. Mit einem + kann eine Hierarchie-Ebene als Wildcard gesetzt werden.

Zugrunde liegt das klassische Observer Pattern dh die publisher müssen nichts über die die subscriber wissen und umgekehrt. Verwendet werden push-updates notification

Üblicherweise baut man für die subscriber asynchrone callbacks, weil man ja nicht weiß, wann eine Nachricht daherkommt.

# Broker:

Das System benötigt einen Broker. Am gebräuchlichsten für MQTT ist der freie mosquitto broker, der sich unter Linux leicht einrichten lässt und auch perfekt auf „schwacher“ Hardware wie einem Raspberry läuft (er empfängt und verteilt ja nur die Daten). Installation ist sehr einfach, mit einem apt-get install mosquitto-server ist erledigt.  
Man kann ihn einfach mit ACLs (access control lists) so einrichten, dass nur gewisse accounts mit Username und Kennwort darauf zugreifen können und zB mit SSL absichern, so dass die Daten verschlüsselt übertragen werden.

Im Internet gibt es auch Test-Broker, die man verwenden kann (die sehen dann aber die gesendeten Daten zb IOT.eclipse.org) oder kommerzielle Broker (zb HiveMQ)

Bibliotheken für alle möglichen Programmeirsprachen gibt es von **paho**

Beispiel zB ein Arduino als client

Man verwendet die Bibliothek

#include <PubSubClient.h> // MQTT Bibliothek

legt einen Client-Object an

PubSubClient client(meineIPAdresse);

Und verwendet die passenden Methoden je nach Sender oder Empfänger

client.setServer(ip und port des Brokers) , client.connect, client.subscribe (topic), client.publish (topic)…

wichtig ist client.setCallback(meinecallbackfunktion);

die Callbackfunktion muss man selber schreiben, dort steht drinnen was passieren soll, wann immer der Broker eine Nachricht published. Die Funktion erhält das topic und den payload der Nachricht

Zum Testen der Clients kann man grafische tools nehmen MQQT, fx oder node-red (eine Erweiterung von Visual Studio Code mit der man über den Browser sehr einfach, grafisch viele Komponenten des IoT zusammenbauen und testen kann – ist wie scratch fürs IoT)

Einfach geht die mqtt Verbindung mit Java Script zb bei node.js

var mqtt = require('mqtt');

var client = mqtt.connect('mqtt://BrokerIP:1883');

und den event-Handlern zB für den Verbindungsaufbau

client.on('connect', function () { hier subscriben oder publishen, also client.subscribe oder client.publish je nachdem was man will});

oder den Erhalt einer Message

client.on('message', function (topic, message) {….});

Elegant ist auch die Verwendung von websockets, wenn man mit html und js arbeitet.   
ZB IOT.eclipse.org kann das. Der Broker kann dann in Echtzeit seine responses senden, wenn eine Nachricht vorhanden ist, und ist nicht auf ein request vom client angewiesen.

Es gibt Bibliotheken für Java, C#, C++, python, js usw. (sind fast alle alle von paho!!!)

Der Ablauf schaut aber immer ähnlich aus:

* client Objekt anlegen
* connect Methode zum Broker (Parameter: Broker, port und eindeutige userID)
* publish und subscribe Methoden mit topics als Parameter
* callback für das subscriben
* close

Im Vergleich dazu könnte man den ganzen Datenverkehr auch mit sockets realisieren (hat dann aber vile Arbeit)

Server als listener – die Clients melden sich mit Rolle als publisher oder subscriber an.   
Kommunikation entspricht dann einem chat Protokoll sollte Flexibilität ermöglichen (XML, json,…), so dass die Nachrichten nach Sender und Empfänger gefiltert gesammelt und weiterverteilt werden könnnen (also eigentlich lauter eigene chatrooms).  
Im OSI Modell ist man bei 3 und 4 für die reine Datenübertragung und 6 bzw 7 für die Auswertung der Datenpakete. Security würde dann ebenfalls über TLS ermöglicht werden.

Beispiel für einen MQTT subscriber, der über websockets zugreift

/\*

\* MQTT-WebClient example for Web-IO 4.0

\*/

var hostname = "ws://iot.eclipse.org/ws";

//erzeugt einen zufällige ID über einen Namen+Zeitstempel

var clientId = "obghtlvillach";

clientId += new Date().getUTCMilliseconds();

mqttClient = new Paho.MQTT.Client(hostname, clientId);

//asynchronen callback erzeugen

mqttClient.onMessageArrived = MessageArrived;

mqttClient.onConnectionLost = ConnectionLost;

Connect();

/\*Initiates a connection to the MQTT broker\*/

function Connect(){

mqttClient.connect({

onSuccess: Connected,

onFailure: ConnectionFailed,

keepAliveInterval: 10

});

}

/\*Callback for successful MQTT connection \*/

function Connected() {

console.log("Connected");

mqttClient.subscribe("htlvillach/OBG/temp");

}

/\*Callback for failed connection\*/

function ConnectionFailed(res) {

console.log("Connect failed:" + res.errorMessage);

}

/\*Callback for lost connection\*/

function ConnectionLost(res) {

if (res.errorCode != 0) {

console.log("Connection lost:" + res.errorMessage);

Connect();

}

}

/\*Callback for incoming message processing \*/

function MessageArrived(message) {

document.getElementById("Chat").innerHTML = message.payloadString;

}