Schlanke Kommunikation

MQTT – Universelles Protokoll für den Datenaustausch

Für den Datenaustausch zwischen Geräten gibt es eine Vielzahl von Protokollen. Im rasant wachsenden Internet der Dinge (IoT) wird ein Protokoll aufgrund seiner besonders für energiesparende Sensoren geeigneten Eigenschaften häufig genutzt: MQTT. Wie das Protokoll aufgebaut ist, welche Möglichkeiten es bietet und wie man es in der Praxis anwendet, zeigen wir in diesem Grundlagenbeitrag.

Internet der Dinge – IoT

Das Internet der Dinge lässt sich vielfältig beschreiben. An dieser Stelle soll die Definition aus Wikipedia [1] verwendet werden: „Das Internet der Dinge (englisch: Internet of Things: IoT) ist ein Sammelbegriff für Technologien einer globalen Infrastruktur der Informationsgesellschaften, die es ermöglicht, physische und virtuelle Gegenstände miteinander zu vernetzen und sie durch Informations- und Kommunikationstechniken zusammenarbeiten zu lassen. Ziel des Internet der Dinge ist es, relevante Informationen aus der realen Welt automatisch zu erfassen, miteinander zu verknüpfen und im Netzwerk verfügbar zu machen."

Während der Zugang von Menschen zum Internet zwischen 2005 bis heute annähernd linear von 1 Milliarde auf 4 Milliarden Menschen anwuchs, ist die Zahl der über das Internet vernetzten Geräte durch ein exponentielles Wachstum gekennzeichnet. 2015 tauschten noch ca. 15 Milliarden Dinge über das Internet Informationen aus, 2020 bereits 31 Milliarden und 2025 wird mit geschätzt 75 Milliarden kommunizierender Dinge gerechnet (Bild 1).

Kommunikation im IoT

Eine wesentliche Grundanforderung an die Kommunikation im IoT ist deren Drahtlosigkeit bei sehr geringem Energiebedarf. Nur so ist es möglich, Sensoren über zehn Jahre und mehr aus Batterien oder gar mittels Energy Harvesting autark aus der Umgebung zu versorgen. Man kann sich vorstellen, dass eine derartige energetische Genügsamkeit nur kurze Botschaften zulässt, die zudem mit äußerst geringer Sendeleistung vom „Ding" abgestrahlt werden. Geringe Sendeleistung bedeutet aber wiederum eingeschränkte Reichweiten, was Vernetzungsarchitekturen mit lokalen Empfangsknoten (Konzentratoren) erforderlich macht.

Bei diesen Konzentratoren laden die Sensoren ihre verschlüsselte Datenfracht ab, in der Erwartung, dass sich der Konzentrator um die Weiterleitung in Richtung des Zielempfängers kümmert. Als Beispiel dafür sei LoRaWAN (Long Range Wide Area Network, mehr dazu unter [2] und [3]) genannt. Hier ist die Grundlage eine sternförmige Netzarchitektur (Bild 2), in der die Objekte leistungsarm und funkbasiert (gestrichelte Pfeile) mit Gateways kommunizieren, welche die Nutzdaten IP-basiert in der Regel leitungsgebunden an Netzwerkserver überträgt. Diese reichen die Daten an Applikationsserver (Anwendungsserver) weiter, wo sie entschlüsselt und für den Anwender zur Nutzung aufbereitet werden.

MQTT – das Standardprotokoll im IoT

Im IoT und bei der Kommunikation unter Maschinen (Machine to Machine, M2M) treten oft unstabile Übertragungsverhältnisse auf, die durch elektromagnetische Störungen, lange Signallaufzeiten und Verzögerungen (Latenzen) oder zeitweise Nichtverfügbarkeit der Verbindung gekennzeichnet sind. Trotzdem muss die Kommunikation auch unter solch widrigen Verhältnissen zuverlässig erfolgen.

MQTT (Message Queueing Telemetry Transport) ist die inzwischen als offener Standard anerkannte Methode, um Meldungen mit minimalem Verwaltungszusatzaufwand (Protocol Overhead) und äußerst energieeffizient über schmalbandige Übertragungskanäle zu transportieren.

MQTT beruht auf IBM MQ, einer von IBM entwickelten Software mit auf Unternehmen abgestimmten Messaging-Funktionen, die Daten optimal und sicher zwischen Anwendungen übertragen. Wenn eine Nachricht nicht sofort übermittelt werden kann, speichert MQ sie so lange in einer Warteschlange (Message Queue), bis die Übermittlung erfolgen kann. So stellt IBM MQ sicher, dass eine Nachricht niemals verloren geht.

Als Andy Stanford-Clark (IBM) und Arlen Nipper (Cirrus Link) 1999 ihr Protokoll MQTT schufen, wollten sie Monitoringdaten von Ölpipelines aus abgelegenen Regionen via Satelliten zu den Betreibern übermitteln. Das dabei entstandene Protokoll benannten sie nach IBM MQ ergänzt um TT als Hinweis auf den Telemetrietransport der Daten (griechisch: Telemetrie = Fernmessung). MQTT wurde 2010 von IBM öffentlich freigegeben und erwies sich Jahre später als äußerst geeignet für das aufkommende IoT.

MQTT in der Übersicht

Damit der Nutzdatenanteil einer Meldung möglichst groß ist, müssen die im Meldungskopf enthaltenen Zusatzinformationen (Protocol Overhead) so knapp wie möglich gehalten werden. Eine sogenannte Client-Broker-Topologie unterstützt diese Forderung. Was bedeutet das?

Schauen wir uns Bild 3 näher an. Wir sehen in der Mitte den mit MQTT-Broker (englisch: broker = Makler) bezeichneten roten Kreis, der von MQTT-Clients (Klienten) umgeben ist. Der grüne, auch als Publisher bezeichnete MQTT-Client veröffentlicht eine Meldung (englisch: to publish = veröffentlichen), indem er sie dem MQTT-Broker schickt.

Der Broker weiß nun, welche Klienten an Meldungen dieses Publishers interessiert sind, weil diese sie abonniert haben und deshalb als Subscriber bezeichnet werden (englisch: to subscribe = abonnieren). Aus diesem Grund benötigt ein Publisher keine Informationen über die Abonnenten seiner Nachrichten. Er muss nur den Kontakt zum Broker herstellen können, um dort seine Meldung abliefern zu können. Dieser kümmert sich um die Verteilung der Meldungen an die Abonnenten in Form von Push-Nachrichten. Der Subscriber muss also nicht regelmäßig beim Broker anfragen, ob eine neue Nachricht zu dem ihn interessierenden Thema vorliegt (polling), sondern erhält sie unaufgefordert zugestellt.

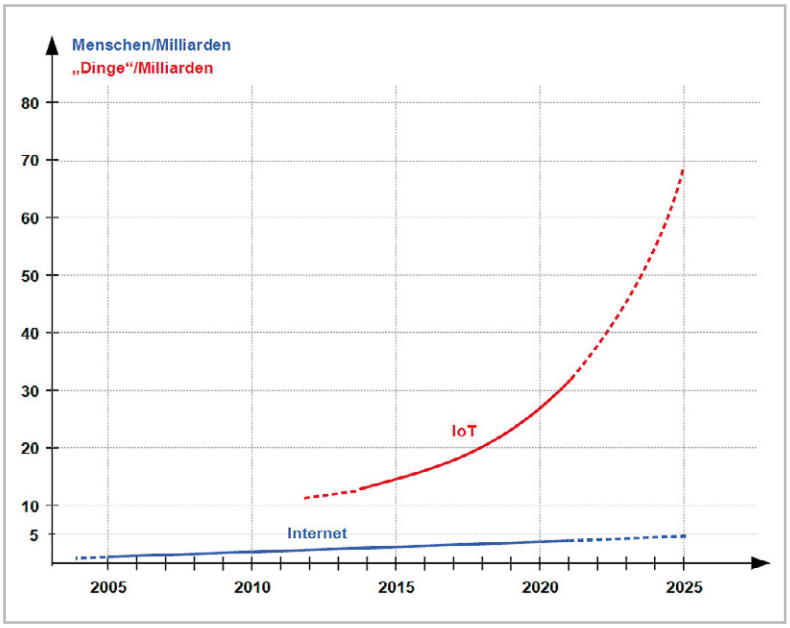

Bild 1: Während die Zahl der Menschen mit Internetzugang durch die Erdbevölkerung begrenzt ist, gilt dies nicht für die Zahl der „Dinge" im IoT.

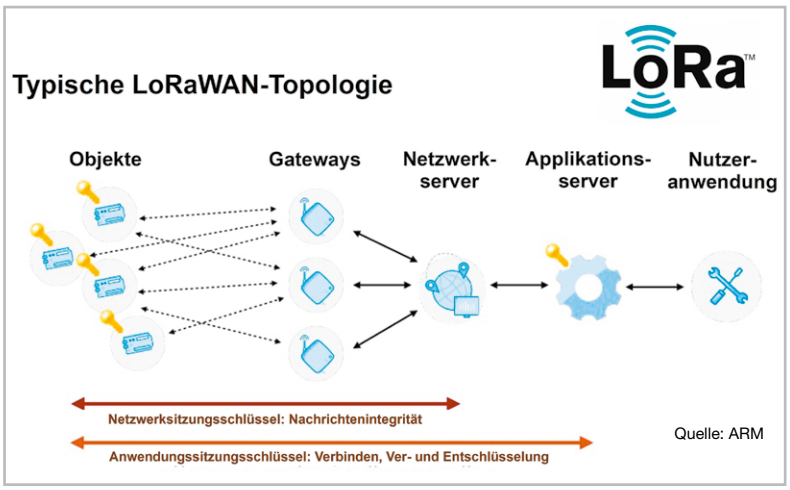

Bild 2: In LoRaWAN-Topologien sammeln Gateways die Meldungen der Clients in ihrem Umfeld.

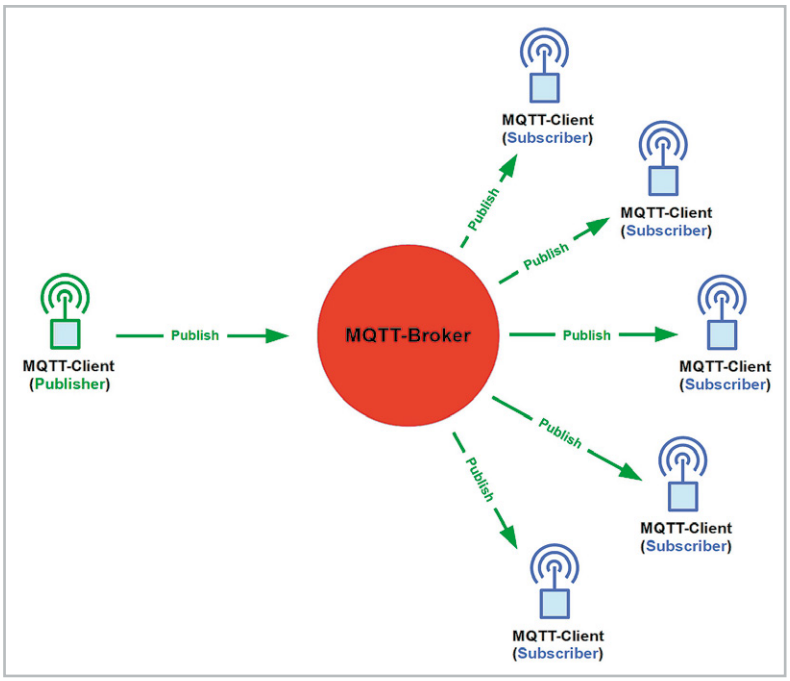

Bild 3: In Broker-Clients-Topologien spielt der Broker eine zentrale Rolle.

Das in MQTT umgesetzte Publisher/Subscriber-System (pub/sub) entkoppelt somit Datenlieferant und Datenempfänger räumlich und zeitlich und stellt besonders geringe Anforderungen an die Rechenleistung der Clients.

In Wikipedia [4] kann man lesen: „Interessant ist, dass ein MQTT-Server (Broker) die gesamte Datenlage seiner Kommunikationspartner hält und so als Zustands-Datenbank benutzt werden kann. So ist es möglich, kleine nicht performante MQTT-Geräte mit einem MQTT-Broker zu verbinden, wobei die Geräte Daten einsammeln und/oder Befehle entgegennehmen, während ein komplexes Lagebild nur auf dem MQTT-Broker entsteht und hier ... ausgewertet werden kann."

Kommunikation per MQTT

Wie die Kommunikation zwischen dem Makler (Broker) und seinen Klienten (Publisher und Subscriber) abläuft, soll nachfolgend elementar beschrieben werden. Dabei wird der Zeichenvorrat UTF-8 verwendet (ein Byte ermöglicht die Darstellung von $2^8 = 256$ Zeichen, von denen die ersten 128 identisch mit dem ASCII-Code sind). Umlaute und andere Sonderzeichen sind nicht erlaubt.

CONNECT

Um mit dem Broker in Verbindung treten zu können, muss sich ein Client zunächst bei ihm anmelden. Dazu dient die CONNECT-Meldung, in welcher sich der Client mit einer „ClientID" identifiziert und mit „Username" und „Password" authentifiziert. Einige weitere wie „Clean Session", „Will Message" und „Keep Alive" sind ebenfalls enthalten. Hat der Broker die CONNECT-Meldung erhalten, quittiert er sie mit einer CONNACK-Meldung (CONNection ACKnowledge = Verbindungsbestätigung). Nun kennen sich Client und Broker und wissen, dass eine funktionierende bidirektionale Verbindung zwischen ihnen besteht.

PUBLISH

Wenn Broker und Client verbunden (connected) sind, kann die Veröffentlichung (Publishing) von Meldungen (Messages) erfolgen. Der wichtigste Bestandteil der PUBLISH-Meldung ist das „Topic" (Thema), ein hierarchisch aufgebauter Textstring. Beispiele für ein Topic können

Haus/Erdgeschoss/Wohnzimmer/Temperatur

oder

Haus/Keller/Fitnessraum/Luftfeuchtigkeit

sein.

Als Trennzeichen zwischen den Hierarchieebenen dient der Schrägstrich (Slash) /. Hinzu kommt die **QoS**-Ziffer 0, 1 oder 2, welche die Erwartung an die Zuverlässigkeit der Übertragung kennzeichnet (QoS: Quality of Service = Dienstgüte).

QoS=0 steht für die Erwartung, dass die Nachricht höchstens einmal beim Broker abgeliefert wird. Das bedeutet, dass der publizierende Client nach dem Absenden seiner Meldung keine Rückmeldung über den Eingang seiner Nachricht erwartet und erhält. Im Englischen wird deshalb eine Message mit QoS=0 als „shoot and forget" (abschießen und vergessen) charakterisiert.

QoS=1 will sicherstellen, dass die Nachricht wenigstens einmal beim Broker eintrifft. Dazu muss der Broker dem Client eine Eingangsbestätigung schicken. Kommt diese z. B. wegen schlechter Übertragungsverhältnisse nicht innerhalb einer bestimmten Zeit (Time-out) an, sendet der Client erneut seine Message mit QoS=1. Irgendwann erhält der Client seine Eingangsbestätigung und die Sendung dieser Nachricht ist für ihn erfolgreich abgeschlossen.

QoS=2 stellt die zeitraubendste, aber sicherste Kommunikation zwischen Client und Broker dar. Allerdings garantiert sie, dass die Message genau einmal beim Broker eintrifft. Dafür quittiert der Broker den Eingang der Meldung und der Client bestätigt den Erhalt der Quittung, worauf der Broker die Beendigung der Kommunikation zurückmeldet. So werden insgesamt vier Meldungen ausgetauscht, was das Datenaufkommen erhöht, die Kommunikation verlangsamt und den Energieverbrauch beim Client erhöht.

Jede Meldung ist sinnlos, wenn sie keine „Payload" (Nutzdaten) enthält. Darunter versteht man die eigentliche Botschaft, deren Übermittlung durch alle weiteren, dem Ablauf der Übertragung dienenden Daten (Metadaten) der Message erst ermöglicht wird. Dabei ist das Format der Nutzdaten unerheblich: Es können (un)verschlüsselter Text, Bilder oder

C:mosquitto

C:\Program Files\mosquitto

Name	Größe Aut...	Geändert am	Typ	Attribute
..		26.10.2020 16:17...	Dateiordner	
aclfile.example	230 Byte(s)	19.08.2020 14:55...	EXAMPLE-Datei	A
ChangeLog.txt	102 kB	19.08.2020 14:55...	Textdokument	A
edl-v10	2 kB	19.08.2020 14:55...	Datei	A
epl-v10	12 kB	19.08.2020 14:55...	Datei	A
libcrypto-1_1-x64.dll	3,25 MB	23.09.2020 05:50...	Anwendungserwe...	A
libssl-1_1-x64.dll	667 kB	23.09.2020 05:50...	Anwendungserwe...	A
mosquitto.conf	44 kB	19.08.2020 14:55...	CONF-Datei	A
mosquitto.dll	82 kB	09.10.2020 16:34...	Anwendungserwe...	A
mosquitto.exe	303 kB	09.10.2020 16:34...	Anwendung	A
mosquitto_passwd.exe	20 kB	09.10.2020 16:33...	Anwendung	A
mosquitto_pub.exe	47 kB	09.10.2020 16:34...	Anwendung	A
mosquitto_rr.exe	46 kB	09.10.2020 16:34...	Anwendung	A
mosquitto_sub.exe	48 kB	09.10.2020 16:34...	Anwendung	A
mosquittopp.dll	18 kB	09.10.2020 16:34...	Anwendungserwe...	A
pwfile.example	355 Byte(s)	19.08.2020 14:55...	EXAMPLE-Datei	A
readme.md	4 kB	19.08.2020 14:55...	MD-Datei	A
readme-windows.txt	3 kB	19.08.2020 14:55...	Textdokument	A
Uninstall.exe	65 kB	26.10.2020 16:17...	Anwendung	A

Bild 4: Programmverzeichnis einer installierten Mosquitto-Version 1.6.12a für Windows 10

andere beliebige Binärdaten sein. Wegen dieser Formatunabhängigkeit der Nutzlast wird MQTT auch als „Data Agnostic“ bezeichnet.

SUBSCRIBE

Das Publishing einer Nachricht ist nur dann sinnvoll, wenn es Abnehmer-Clients dafür gibt. Ein solcher meldet sich beim Broker durch eine Subscribe-Meldung an. Diese enthält das gewünschte „Topic“, z. B. „Haus/Erdgeschoss/Wohnzimmer/Temperatur“. Wenn der Broker den Eingang einer solchen Meldung verzeichnet, quittiert er sie dem subscribierenden Client mit einer SUBACK-Meldung (SUBscribe ACKnowledge = Abonnement bestätigt).

Ab nun erhält der Subscriber-Client alle Meldungen zu dem angemeldeten Thema unaufgefordert durch Push-Nachrichten zugestellt. Entsprechend meldet sich ein Client als Bezieher von Meldungen durch einen Austausch von UNSUBSCRIBE/UNSUBACK-Meldungen beim Broker ab.

Praxis

Das bisher Geschilderte kann durch einige einfache Experimente ohne Hardware am Computer veranschaulicht werden.

Experiment 1

Steht nur ein PC zur Verfügung, müssen wir Broker, Subscriber und Publisher auf diesem gemeinsam abbilden. Dazu installieren wir die kostenlose Open-Source-Broker-Software namens Mosquitto, die unter [5] für Plattformen wie Windows, Mac und verschiedene Linux-Distributionen (u. a. auch für den Raspberry Pi) erhältlich ist. Wir beschreiben nachfolgend am Beispiel von Windows die Funktionalität.

Nach der Installation sollte das Verzeichnis

C:\Program Files\mosquitto

in etwa wie in Bild 4 – in diesem Fall der Windows-10-Rechner des Autors – aussehen.

Wir öffnen jetzt eine Kommando-Konsole (Eingabeaufforderungsfenster) unseres PCs. Die Konsole erreichen wir, indem wir über die Tastenkombination Windows-R das Ausführen-Fenster aktivieren und dort „cmd“ eingeben.

Wir wechseln mit

cd c:\Program Files\mosquitto

in das Mosquitto-Programmverzeichnis und starten den Broker mit

mosquitto.exe

Wir öffnen ein weiteres Konsolen-Fenster (Subscriber-Konsole), indem wir wieder in das Mosquitto-Programmverzeichnis wechseln und den Subscriber unter Nennung des abonnierten Topics

```
mosquitto_sub -t Haus/Erdgeschoss/↩
Wohnzimmer/Temperatur
```

starten.

Mit einem blinkenden Unterstrich signalisiert nun der Subscriber, dass er auf Messages zum genannten Topic wartet, nämlich auf Temperaturmesswerte aus dem im Erdgeschoss des Hauses gelegenen Wohnzimmer.

In einer dritten Konsole (Publisher-Konsole) setzen wir jetzt eine Message mit Temperaturwert mit der Befehlszeile

```
mosquitto_pub -t Haus/Erdgeschoss/↩
Wohnzimmer/Temperatur -m ↩
"Wohnzimmertemperatur 21.6 C"
```

ab.

Sofort erscheint diese Meldung im Subscriber-Fenster. Das können wir mit geänderten Werten beliebig oft wiederholen. Ändern wir etwas an den Topics, sodass sie nicht mehr übereinstimmen, erfolgt keine Übertragung der Meldungen. Bild 5 fasst das Gesagte zusammen.

Die ersten drei Messages des Publishers kommen beim Subscriber wegen der gleichen Topics an, nicht dagegen die vierte Message, weil sie ein anderes Topic hat.

Hier kann man sich helfen, indem man den Subscriber mit dem Befehl

```
mosquitto_sub -t Haus/Erdgeschoss/#
```

startet.

Das #-Zeichen ersetzt dabei alle Hierarchieebenen unterhalb von mosquitto_sub -t Haus/Erdgeschoss, wodurch auch die Message eines Publishers mit dem Topic -t Haus/Erdgeschoss/Kinderzimmer/Temperatur -m „Kinderzimmertemperatur 19.5 C“ vom Subscriber übernommen wird.

Bild 5: Zusammenspiel zwischen Client (Publisher) und Client (Subscriber)

Mit einer derartigen sogenannten Wildcard-Konstruktion muss also nur bei Publisher und Subscriber die Hierarchie bis zum # übereinstimmen. Beliebig viele weitere Hierarchieebenen werden durch # ersetzt. Die Auswirkungen sind in Bild 6 nachzuverfolgen.

Das bisher Geschilderte können wir auch auf dem Raspberry Pi 4B (oder einer seiner Vorgängertypen) realisieren. Dazu installieren wir die Raspbian-Version von Mosquitto [4] und alles Weitere läuft ganz dem Windows-Beispiel entsprechend. Das Ergebnis zeigt Bild 7.

Experiment 2

Etwas realitätsnäher ist es, Subscriber- und Publisher-Clients auf getrennten Rechnern laufen zu lassen. Dazu wurde ein Raspberry Pi 4B (im Folgenden auch Raspi genannt) mit dem Betriebssystem Raspbian aufgesetzt und über seine eingebaute WLAN-Schnittstelle unter der IP-Adresse 192.168.178.73 drahtlos in das WLAN des Autors eingebunden. Auf diesem Raspi wird Node-RED eingerichtet. Dabei handelt es sich um ein von IBM entwickeltes grafisches Entwicklungssystem, das mit einem Baukasten voller Funktionsbausteine ausgestattet ist.

Einen dreiteiligen Einführungskurs zu Node-RED und der Installation auf einem Raspberry Pi finden Sie im ELVjournal unter [6].

Der Anwender braucht in Node-RED nur die erforderlichen Bausteine auf den Konfigurationsbildschirm zu ziehen, einige Einstellungen an ihnen vorzunehmen und sie zu verbinden. Der sich ergebende „Flow“ repräsentiert die Problemlösung. Der zweite Rechner ist wie bisher der Windows-10-PC mit der Netzwerkadresse 192.168.178.33. Nun ist es mit geeigneter Software möglich, ohne eine Zeile Programmcode zu schreiben, Messages über die WLAN-Schnittstelle zu übertragen.

Dazu installieren wir auf dem PC zunächst das kostenlose Programm MQTT.fx, ein visuelles Debugging-Tool für MQTT-Kommunikation.

MQTT.fx ist eine Desktopanwendung zur komfortableren Kommunikation mit MQTT-Brokern wie HiveMQ oder Mosquitto, als sie über die Standardkonsole möglich ist. Dabei nutzt MQTT.fx die Standardfunktionen der Broker, beim mosquitto-Broker z. B. mosquitto_pub und mosquitto_sub. Wir können den Installationsfile für einen 64-bit-Windows-10-Rechner direkt von der Homepage des Entwicklers Jens Deters laden [7].

Nach der Installation öffnen wir MQTT.fx und nehmen unter Extras → Edit Connection Profiles in dem sich öffnenden Editorfenster für die Verbindung einige Einstellungen vor (Bild 8).

Die Eingabe eines beliebigen Profilnamens (Profile Name), Wahl des Profiltyps (Profile Type) als MQTT Broker, Eintragung der Broker-Adresse (Broker Address) 192.168.178.73 und des Broker-Ports 1883 bestätigen wir mit OK und finden uns im Eingangsbildschirm von MQTT.fx wieder.

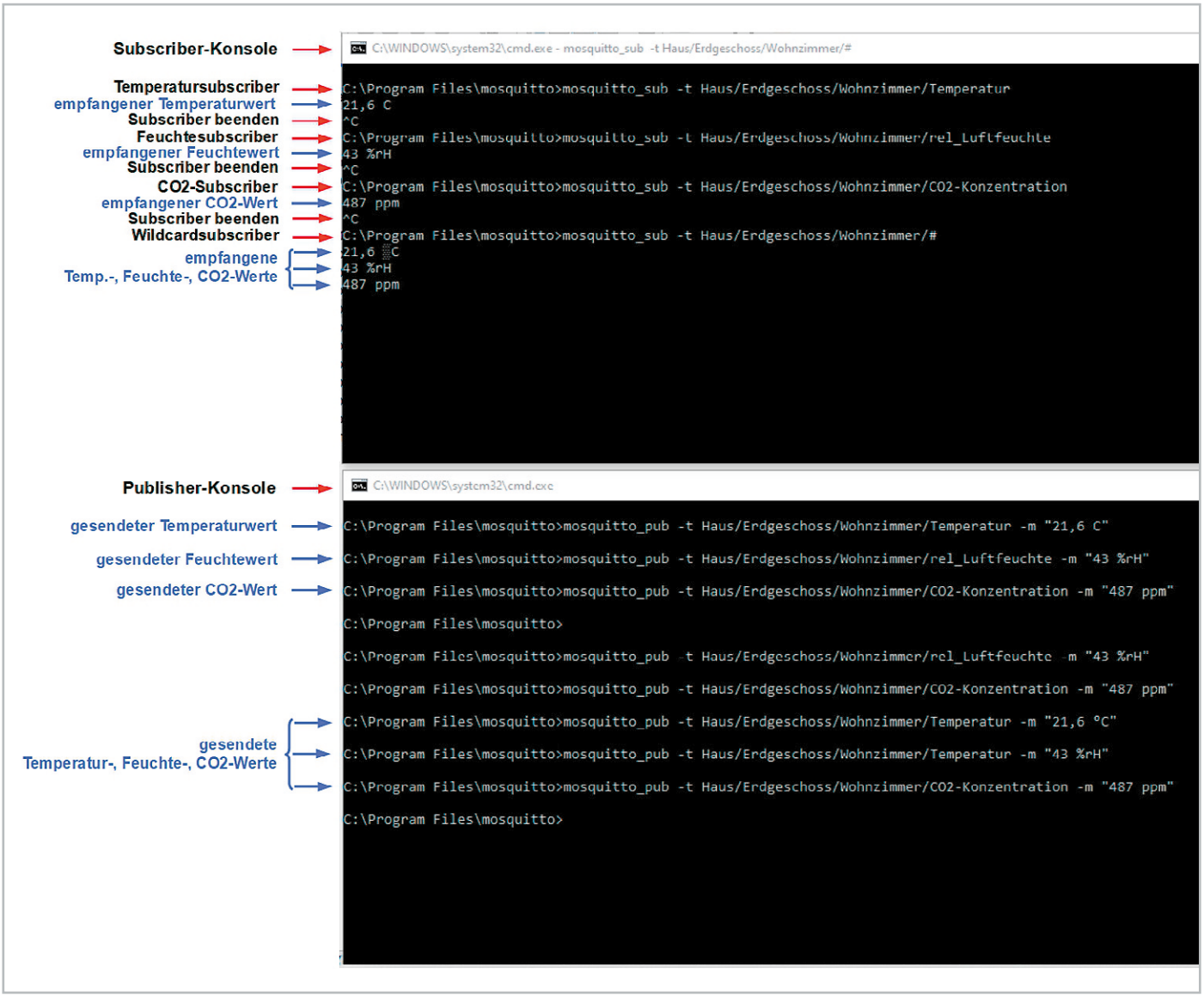

Bild 6: Ein Publisher veröffentlicht Meldungen zu einem bestimmten Topic, die der Broker an darauf abonnierte Subscriber weiterleitet.

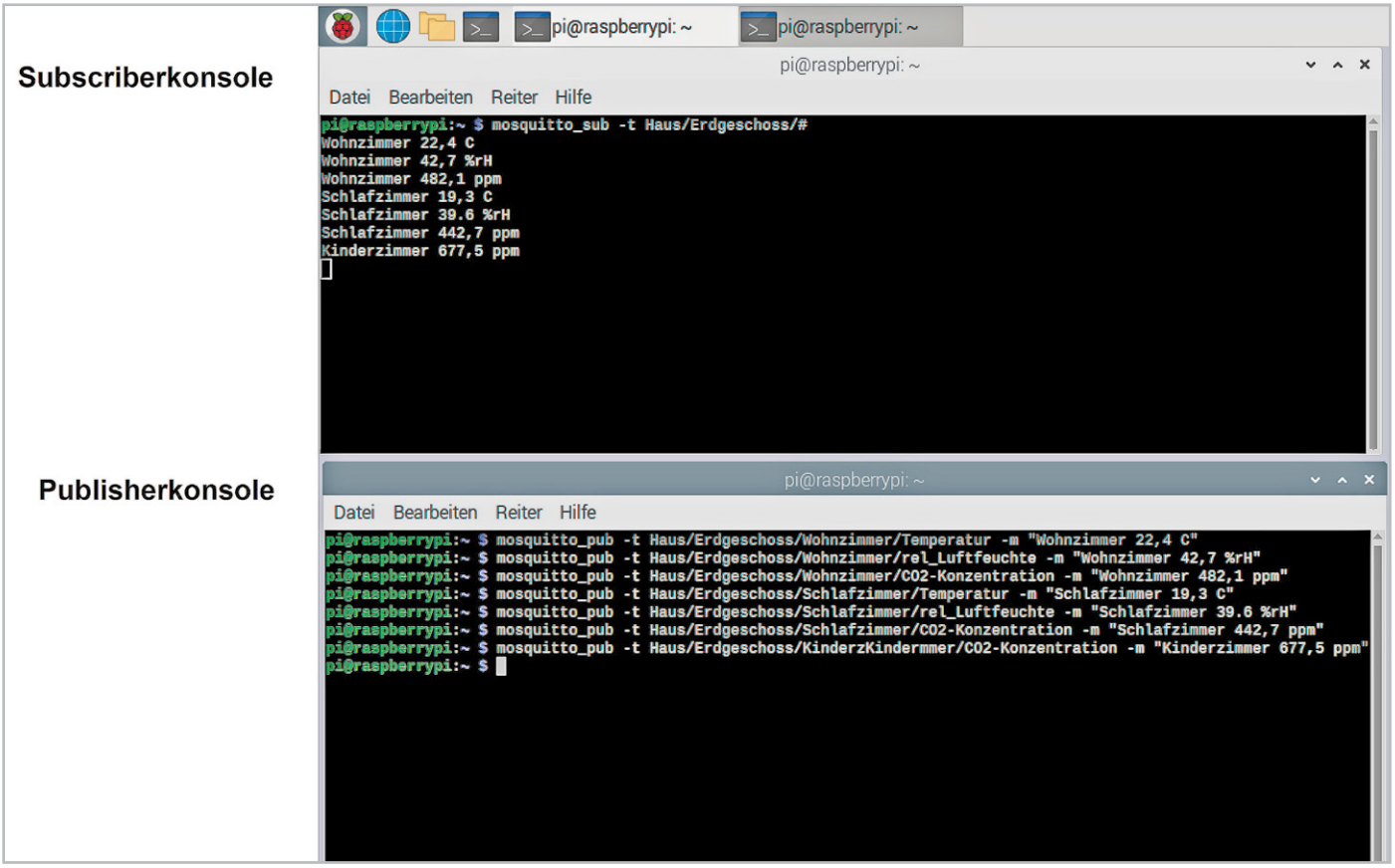

Bild 7: Mosquitto auf zwei Raspberry-Pi-4B-Konsolen

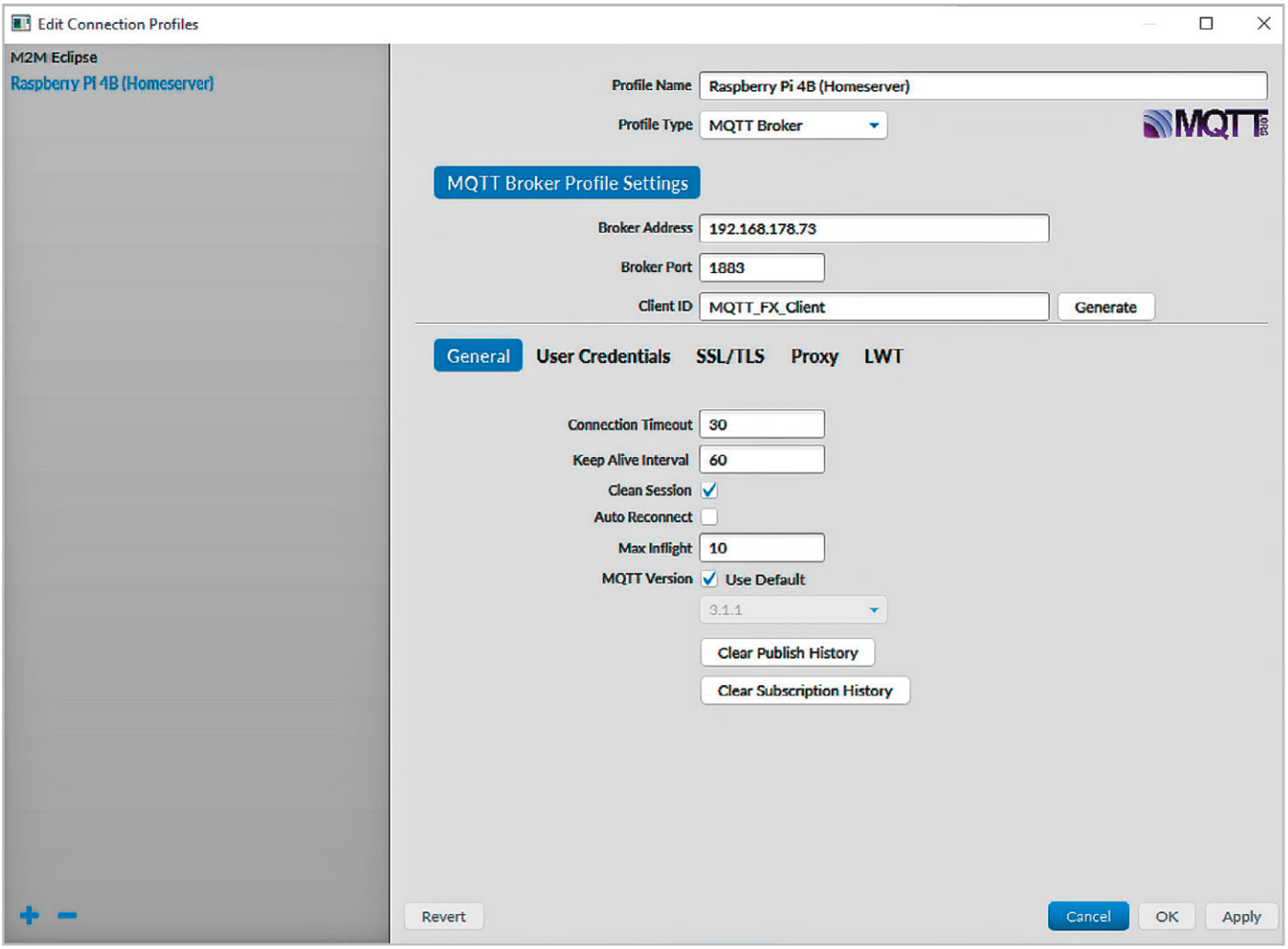

Bild 8: Profileinstellungen in MQTT.fx

Hier stellen wir durch Drücken des Buttons „Connect“ und Eingabe des Topics

Haus/Erdgeschoss/Wohnzimmer/Temperatur

sowie Wahl von QoS 0 die von MQTT.fx weiter benötigten Verbindungdaten bereit (Bild 9).

Nun wenden wir uns wieder dem Raspi zu.

An dieser Stelle soll vorausgesetzt werden, dass Node-RED auf dem Raspi lauffähig ist. Wenn man aus der Benutzeroberfläche des Raspbian-Betriebssystems heraus Node-RED starten will, öffnet sich nicht etwa die Benutzeroberfläche von Node-RED, sondern ein Konsolenbildschirm, dem wir entnehmen können, dass Node-RED aus dem Raspi-Browser heraus durch Eingabe von 192.168.178.73:1880 in die Adresszeile gestartet werden muss (Bild 10).

Ist die Benutzeroberfläche von Node-RED präsent, wird der Node „mqtt in“ aus der Rubrik „network“ der senkrechten Nodeauswahlliste links auf die Arbeitsfläche gezogen und durch zweifachen Linksklick zur Eingabe von Topic „Haus/Erdgeschoss/Wohnzimmer/#“ und QoS-Level 0 geöffnet. Als Output verwenden wir „a String“ und vergeben einen Namen, im Beispiel „Wohnzimmer“ (Bild 11). Mit Klick auf die Schaltfläche „Fertig“ beenden wir die Bearbeitung des Nodes.

Als Nächstes ziehen wir den Node „Debug“ aus der Rubrik „Common“ und verbinden ihn mit dem MQTT-Node, dem wir den Namen „Wohnzimmer“ gegeben haben. Mit Klick auf „Deploy“ wird die Anordnung auf Fehler überprüft und kompiliert.

Im Erfolgsfall leuchtet am oberen Bildschirmrand kurz der Schriftzug „Erfolgreich implementiert“ auf (Bild 12).

Jetzt kommt der Test. Dazu publishen wir in MQTT.fx auf dem Windows-PC eine Nachricht und prüfen, ob diese im Debug-Fenster von Node-RED auf dem Raspi erscheint. Wie Bild 13 beweist, funktioniert das.

Wir haben also MQTT-Messages vom Windows-10-Rechner mithilfe des Programms MQTT.fx unter einem Topic publiziert und der über WLAN angebundene Raspberry Pi 4B hat diese Nachrichten, deren Topic er subscribiert hat, empfangen. Und das ohne jeden Programmcode!

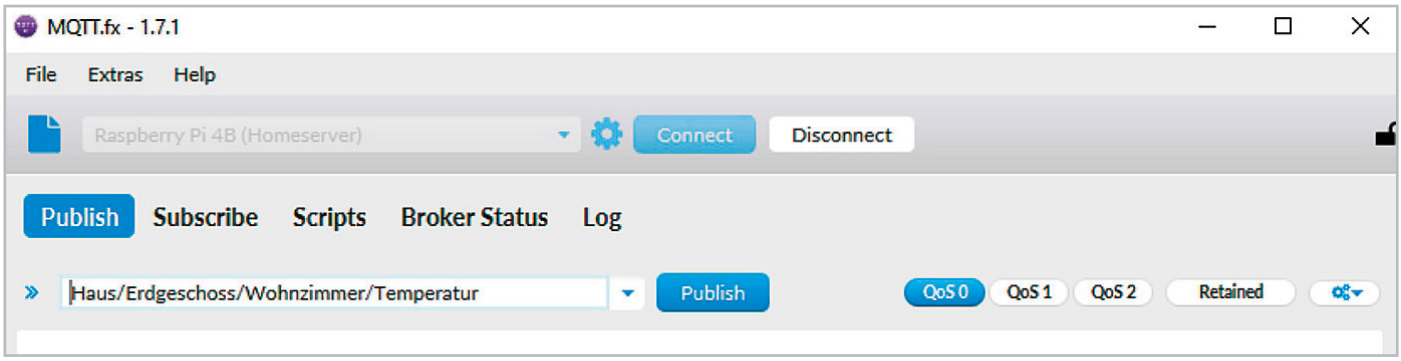

Bild 9: Nach dem Klick auf „Connect“ muss noch das Topic eingeben werden.

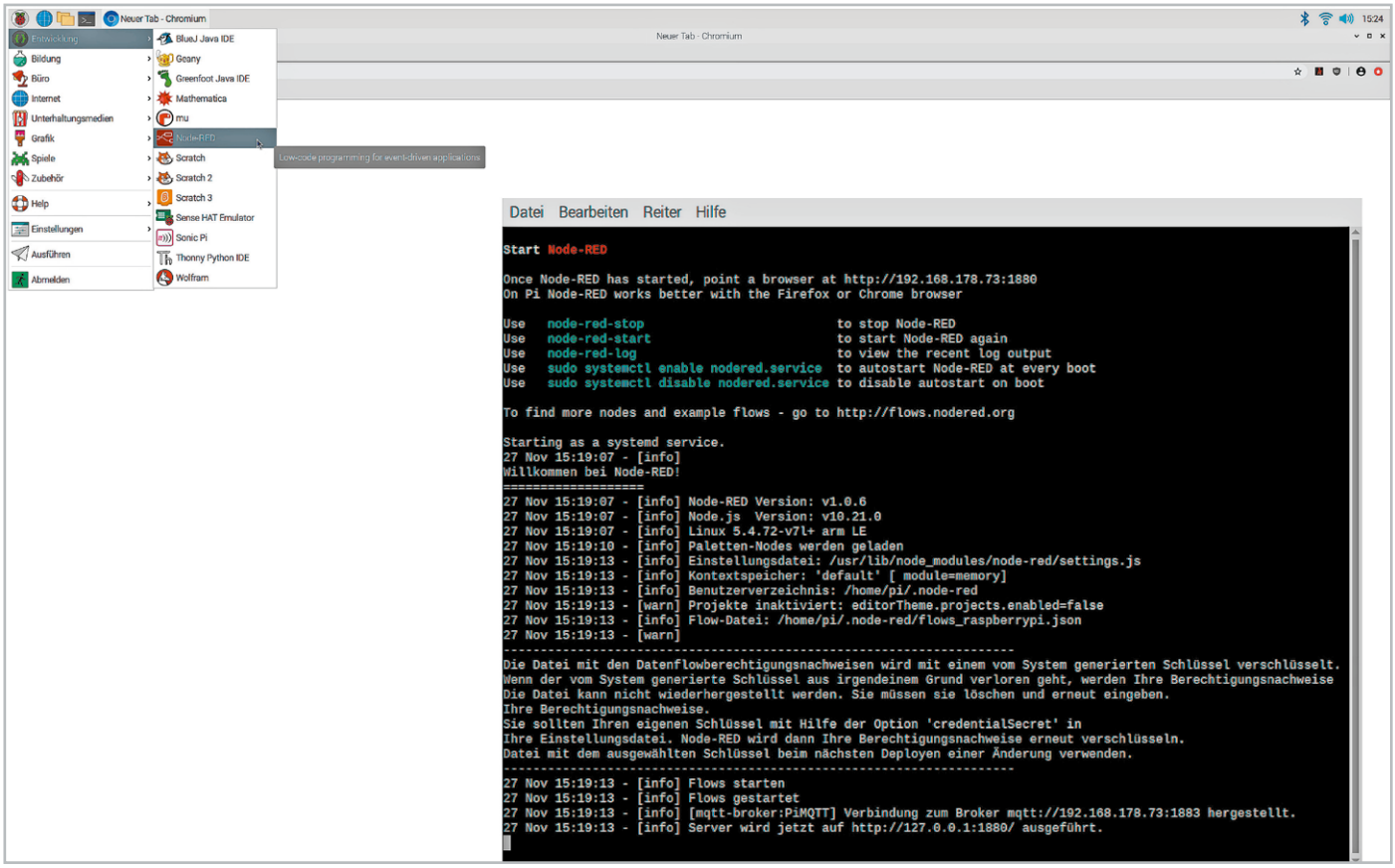

Bild 10: Debug-Ausgaben beim Start von Node-RED

Fazit

Es ist durchaus möglich, in relativ kurzer Zeit ein Grundverständnis für das Protokoll MQTT im IoT und IIoT (Industrial Internet of Things) zu entwickeln. Richtig spannend wird es, wenn man echte „Dinge“ zum Datenaustausch über größere Distanzen bewegen kann. Mit dem Raspberry Pi und den darauf bereits installierten Tools wie Node-RED, der Programmiersprache Python und einer schier unerschöpflichen Flut von teils exzellenten Lernhilfsmitteln und Projektbeschreibungen aus dem Internet kann man das mit der nötigen Beharrlichkeit im Selbststudium schaffen. Tastatur, Maus und Bildschirm sind für den Raspberry nicht zwingend erforderlich, wenn man ihn über das RDP (Remote Desktop Protocol) mit seinem in der Regel ohnehin vorhandenen Windows-PC verbindet. Dann kann es mit einer Anfangsinvestition deutlich unter 100 Euro losgehen. So ist es auch auch bei diesem Projekt der Fall gewesen. ELV

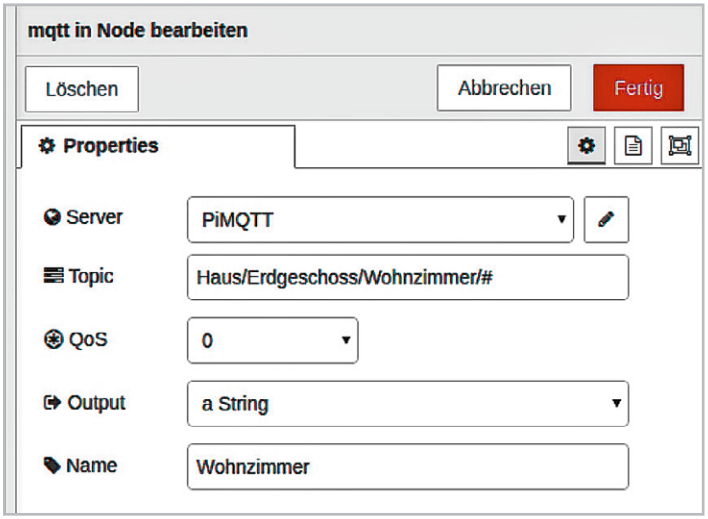

Bild 11: Konfiguration des mqtt in-Nodes

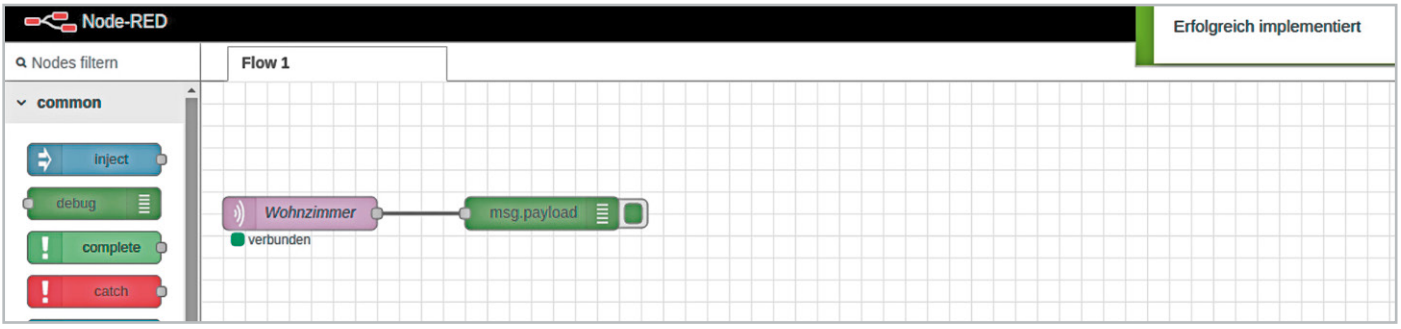

Bild 12: Deployment des Flows

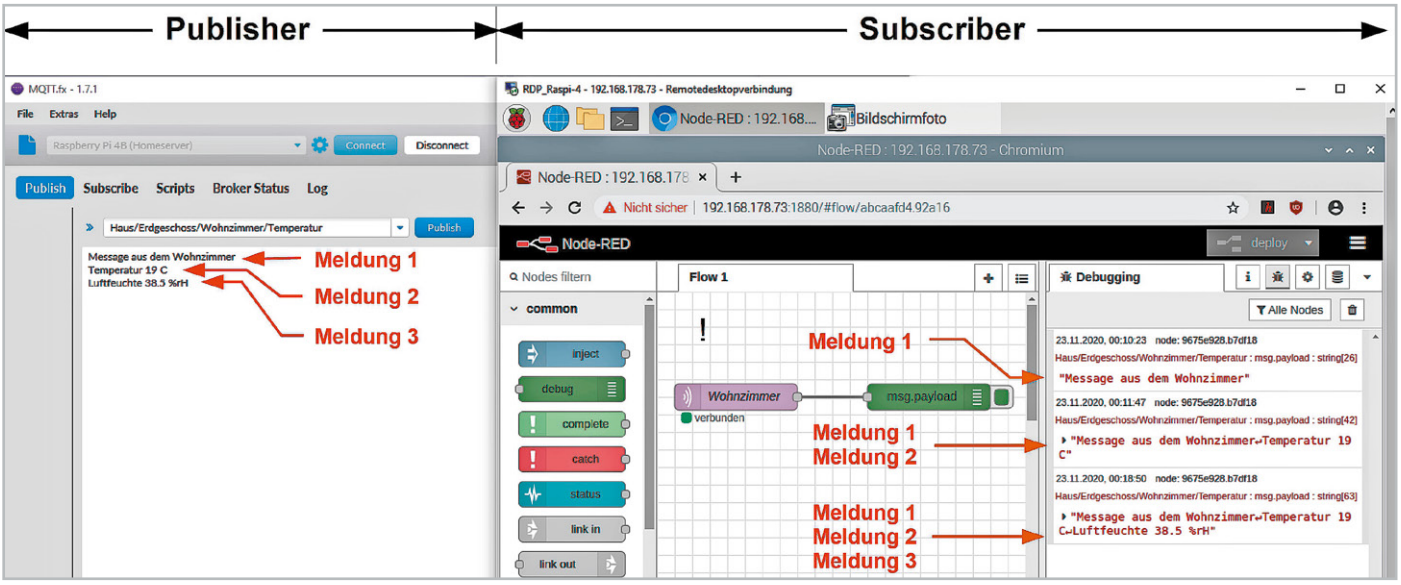

Bild 13: Die Eingabe einer Meldung im Publisher (Node-RED auf Raspberry Pi 4B) erscheint im Subscriber (MQTT.fx auf Windows-Rechner).

Weitere Infos:

[1] Internet der Dinge auf Wikipedia: https://de.wikipedia.org/wiki/Internet_der_Dinge
[2] ELVjournal 1/2021: Energiesparer LPWAN – Low Power Wide Area Networks – Artikel-Nr. 251814
[3] Low Power – Long Range – Die Technologie hinter LoRa und LoRaWAN – Artikel-Nr. 252094
[4] Wikipedia – MQTT: https://de.wikipedia.org/wiki/MQTT
[5] Mosquitto (Eclipse) Download: http://mosquitto.org/download/
[6] ELVjournal-Fachbeitrag Node-RED Einführung (3 Teile): Artikel-Nr. 251410, 251516 und 251603
[7] Download MQTT.fx: http://www.jensd.de/apps/mqttfx/

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links