Fakultät für
Informations-, Medien-
und Elektrotechnik

Masterarbeit Technische Informatik

Automated Management and Performance Analysis of Wi-Fi 6E (6 GHz) Networks for IoT-Systems

Automatisiertes Management und Performance-Analyse von Wi-Fi 6E (6 GHz) Netzwerken für IoT-Systeme

vorgelegt von

Philipp Kalytta

Mat.Nr. 11109927

Erstgutachter: Prof. Dr. Andreas Grebe (Technische Hochschule Köln)

Zweitgutachter: Prof. Dr. René Wörzberger (Technische Hochschule Köln)

Juni 2022

Technology
Arts Sciences
TH Köln

Masterarbeit

Titel: Automatisiertes Management und Performance-Analyse von Wi-Fi 6E (6 GHz) Netzwerken für IoT-Systeme

Gutachter:

- Prof. Dr. Andreas Grebe (TH Köln)
- Prof. Dr. René Wörzberger (TH Köln)

Zusammenfassung: Geräte im Internet of Things (IoT), z.B. Sensoren und Aktoren verfügen über verschiedene Verbindungsmöglichkeiten. Ein verbreiteter Standard zur Kommunikation ist IEEE 802.11 Wireless LAN. Für diesen Standard ist zuletzt die Erweiterung 802.11ax erschienen, die neben neuen Technologien wie OFDMA, BSS coloring u.a. auch die Nutzung des 6 GHz Frequenzbandes ermöglicht. Die Wi-Fi Alliance nennt diese neue Erweiterung Wi-Fi 6E. Die zentrale Verwaltung von IoT-Geräten, die über diese neue Technologie kommunizieren, sowie die Evaluierung der Möglichkeiten und Einschränkungen von Wi-Fi 6E und der bisher dafür verfügbaren Hardware werden im Rahmen dieser Arbeit näher beleuchtet.

Stichwörter: Wi-Fi, WLAN, IEEE 802.11, Data-Rates, 6 GHz, 802.11ax, Wi-Fi 6E, OFDM, OFDMA, Roaming, Device Management, IoT

Datum: 16. Juni 2022

Master’s Thesis

Title: Automated Management and Performance Analysis of Wi-Fi 6E (6 GHz) Networks for IoT-Systems

Reviewers:

- Prof. Dr. Andreas Grebe (TH Köln)
- Prof. Dr. René Wörzberger (TH Köln)

Abstract: Devices on the Internet of Things (IoT), e.g. sensors and actuators, have various connection options. A common standard for communication is IEEE 802.11 Wireless LAN. The latest extension to this standard is 802.11ax, which enables the use of the 6 GHz frequency band in addition to new technologies such as OFDMA, BSS coloring and others. The Wi-Fi Alliance calls this new extension Wi-Fi 6E. The central management of IoT devices that communicate via this new technology as well as the evaluation of the possibilities and limitations of Wi-Fi 6E and the hardware available for it so far are examined in more detail in this work.

Keywords: Wi-Fi, WLAN, IEEE 802.11, Data-Rates, 6 GHz, 802.11ax, Wi-Fi 6E, OFDM, OFDMA, Roaming, Device Management, IoT

Date: June 16th, 2022

Table of Contents

Index of Abbreviations

Abbreviation	Term
ACK	Acknowledgement
AP	Access Point
API	Application Programming Interface
ATIM	Announcement Traffic Indication Message
BPSK	Binary Phase Shift Keying
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
BNetzA	Bundesnetzagentur/Federal Network Agency of Germany
CCK	Complementary Code Keying
CF	Contention Free
CoAP	Constrained Application Protocol
CRC	Cyclic Redundancy Check
CRDA	Central Regulatory Domain Agent
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CTS	Clear to Send
DFS	Dynamic Frequency Selection
DSSS	Direct Sequence Spread Spectrum
EEPROM	Electrically Erasable Programmable Read-only Memory

Abbreviation	Term
EIRP	Equivalent Isotropically Radiated Power
ETSI	European Telecommunications Standards Institute
ESS	Extended Service Set
FCS	Frame Check Sequence
GNU	GNU Project
HE	High Efficiency
HT	High Throughput
IBSS	Independent Basic Service Set
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IFS	Interframe Space
INI-File	Initialization File
IoT	Internet of Things
ISO	International Organization for Standardization
JWT	JSON Web Token
LAN	Local Area Network
LWM2M	Lightweight Machine 2 Machine Protocol
LPI	Low Power Indoor (Devices)
MAC	Media Access Control

Abbreviation	Term
MBSS	Mesh Basic Service Set
MCS	Modulation and Coding Scheme
MIMO	Multiple Input Multiple Output
MQTT	Message Queuing Telemetry Transport
NDP	Null Data Packet
NIC	Network Interface Card
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OUI	Organizationally Unique Identifier
OSI-Model	Open Systems Interconnection Model
PCAP	Packet Capture
PPDU	Physical Protocol Data Unit
PLCP	Physical Layer Convergence Procedure
PSDU	PLCP Service Data Unit
PS	Power Save
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase-Shift Keying
RPC	Remote Procedure Call

Abbreviation	Term
RTS	Request to Send
SSID	Service Set Identifier
STA	Station
TP	Transmit Power
TSFT	Time Synchronization Function Timer
TU	Time Units
VHT	Very High Throughput
VLP	Very Low Power (Devices)
Wi-Fi	Here: Synonym to WLAN, except when referring to the Wi-Fi Alliance
WLAN	Wireless Local Area Network

Glossary

Term	Explanation
Central Regulatory Domain Agent (CRDA)	The CRDA mediates between the kernel and user space through the Netlink interface, ensuring that the regulatory framework in which a WLAN-enabled device operates is met [1]. This process is implemented using the Regulatory Database, which is used by CRDA to comply with the regulatory framework for individual states or territories.
Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA)	A method of avoiding collisions of data transmissions on a carrier medium that is used by several subscribers. The transmission channel is actively monitored by the participants (carrier sense) and if the medium is busy, transmission does not take place. The system always checks whether the medium is free before transmitting.
Equivalent Isotropically Radiated Power (EIRP)	The equivalent isotropic radiated power is the product of the power delivered to the antenna and the antenna gain in a given direction in relation to an isotropic antenna.

Term	Explanation
Hidden-Node-Problem	The Hidden-Node-Problem describes a problem in wireless networks where a station (node) can communicate with the access point, but not directly with other stations. Thus, the data transmission of the other station cannot be detected, which allows the station to communicate with the access point at the same time as the other stations. The interference this creates ensures that the access point does not understand either transmission. The RTS/CTS algorithm in WLAN networks solves this problem by allowing clients to request send authorization before sending.
Media Access Control (MAC)	MAC describes part of the data link layer described in the OSI model (layer 2), which is divided by the IEEE into two sublayers: Media Access Control and, above it, Logical Link Control. The MAC layer controls the physical transmission on a shared transmission medium.
Modulation and Coding Scheme (MCS)	The MCS indexes the data rate used in WLANs. The MCS index can be used together with other parameters (such as the number of spatial streams) to determine the data rate for a WLAN connection.

Term	Explanation
Monitor Mode	Monitor mode is an operating mode for WLAN adapters in which all received network frames are forwarded to the kernel or applications and not only those originally intended for the adapter (destination fields are ignored). Depending on the manufacturer this also applies to corrupted frames, but not every manufacturer forwards these.
Multiple Input Multiple Output (MIMO)	MIMO refers to a method in which multiple transmitting and receiving antennas are used between the participants in a wireless communication. This can significantly increase the data rate if both the transmitter and receiver are capable of MIMO.
Multiple User MIMO (MU-MIMO)	With MU-MIMO a station can transmit to multiple stations simultaneously using multiple antennas, this means the airtime can be used to communicate with multiple participants at the same time.
Orthogonal Frequency Division Multiplexing (OFDM)	Orthogonal frequency division multiplexing (OFDM) is a modulation method that uses multiple orthogonal carriers (zero crossing of neighboring carriers is at the maximum of the carrier). This reduces signal crosstalk compared to non-orthogonal frequency division multiplexing. WLAN uses 48 carriers (+4 pilot carriers) and a carrier signal is usually pre-modulated separately using quadrature amplitude modulation (QAM) (or BPSK/QPSK).

Term	Explanation
Orthogonal Frequency Division Multiple Access (OFDMA)	With OFDMA, the OFDM carriers (see OFDM) are split over more than one user channel. The prerequisite is that bidirectional communication is used and that the channel is measured. Passive measurement means that the transmitter knows the reception quality of the orthogonal carriers to the individual users and the spectral efficiency can be optimized.
Promiscuous Mode	In promiscuous mode a network controller forwards all data to the CPU that is received on the interface, i.e. not only data that is actually intended for its own system. This mode lays the foundation for recording network traffic. For WLAN adapters this mode is not to be confused with Monitor Mode.
Quadrature Amplitude Modulation (QAM)	Quadrature amplitude modulation is a modulation method that combines phase modulation and amplitude modulation. In this process, two carriers with a phase shift are multiplied and added in such a way that the transmit signal is created from them. In WLAN QAM is used in conjunction with OFDM to modulate the individual carriers of OFDM modulation within themselves.

Introduction

In July 2021, the German Federal Network Agency made it possible for everyone to use WLAN[1] devices and applications in the 6 GHz frequency band with a new general allocation. The new 480 MHz spectrum will again expand the WLAN frequency range that can be used by businesses and consumers in order to meet the ever-increasing requirements. The previous frequency bands are already often heavily utilized in densely populated areas. It is to be expected that the other European countries will also make it possible for the general public to use the 6 GHz frequency band in the near future on the basis of the technical guidelines published by ETSI.

The German Federal Network Agency has designated the frequency range approved in Germany in particular for low-power indoor devices (LPI) and very low-power devices (VLP). These are devices with a maximum isotropic radiated power of 200 mW (LPI) or 25 mW (VLP). Especially outdoors, but also indoors, for VLP, an application for the Internet of Things (IoT) is to be expected, since the devices used there are usually optimized for low power consumption.

In the context of the research work by the Computer Networks Research Group at the University of Applied Sciences of Cologne, the new radio network standard 802.11ax of the IEEE (in the following Wi-Fi 6), in particular the part of the standard which refers to the operation in the 6 GHz frequency band (Wi-Fi 6E), is to be considered. Measurement sensors are used (Linux-based), which are to communicate with each other and with a central server over the 6 GHz frequency band via wireless LAN. The management software for this server will also be evaluated as part of this master thesis:

The focus is on the evaluation of two different software solutions for the management of IoT devices in 6 GHz networks as well as on the possibilities for analyzing the quality of service (QoS) and performance and their assessment for these devices.

[1] The terms WLAN (Wireless Local Area Network) and Wi-Fi are not the same. WLAN refers to the wireless technology described by the IEEE 802.11 standard. Wi-Fi is the marketing term used by the Wi-Fi Alliance for devices and networks that use WLAN and are tested for compliance with the standard. They are nevertheless often used synonymously.

1 Problems and Objectives

The problems informing this work can be divided into two groups for which independent objectives apply: The considerations concerning the technical properties of the new Wi-Fi 6E standard can be summarized as one group. The other group comprises the problems concerning the management of devices in networks operating in the 6 GHz band. The problems can therefore be divided as follows:

1. Which software is suitable for managing Wi-Fi 6E-enabled network devices in the IoT environment?
2. How can these devices be automatically provisioned, configured and operated through central management?
3. What can be said about the performance of the currently available hardware for Wi-Fi 6E in this environment?
4. Is Wi-Fi 6E with current hardware suitable for the operation of networked IoT devices with centralized, automatic management?

Special attention is paid to the technical framework of the current Wi-Fi 6E-capable hardware: The associated limitations and possibilities (e.g., channel bandwidths, modulation) are to be tested and evaluated in various measurement scenarios, with the focus on the areas of quality of service (QoS) and performance. The basis for this is the construction of a hardware platform that must be capable of communicating over the 6 GHz Wi-Fi band. This hardware platform will also be used to implement central configuration and operation with the aid of central management software. Two different management software solutions are being evaluated for this purpose. This also requires the design of an additional software architecture to allow the Wi-Fi 6E-enabled devices to communicate with the management software. This will be implemented in a laboratory environment with the available hardware.

2 Basis

This chapter describes the basics of wireless networks based on the IEEE 802.11 standard (also referred to as WLAN, wireless LAN or the marketing term Wi-Fi in the following). The 802.11 standard describes the lowest two layers in the ISO standard Open Systems Interconnection (OSI) Model for the exchange of information in wireless systems: The physical layer (PHY) and the medium access control layer (MAC) [2]. Also described is the general functionality of IoT device management software.

2.1 IEEE 802.11 Wireless LAN

The 802.11 standard specifies the transmission of data via a radio link for local area networks. Usually, the data is exchanged in the next higher OSI layer (layer 3) between two (or more) devices using the Internet Protocol (IP). In wired networks Ethernet is generally used which specifies the two lowest layers of the OSI model. For data transmission in wireless networks the IEEE adopted the first of several standards in 1997, which has been extended several times since. In 2018 the Wi-Fi Alliance, a consortium of organizations including network hardware manufacturers, Internet companies, and mobile network operators [3], introduced marketing terms for the various versions of the standard under the designation Wi-Fi N, where N denotes an ascending version number (e.g. Wi-Fi 5) [4]. The term Wi-Fi refers to certified products of the Wi-Fi Alliance that are 802.11 standard-compliant for the respective standard version. The 802.11 standard has been extended several times, in particular to meet the increased data rate requirements. The original standard specifies for transmission in the 2.4 GHz frequency band with a maximum data rate of 2 Mbit/s gross. As early as 1999, the standard was extended: 802.11a (first extension) allows data rates of up to 54 Mbit/s gross in the 5 GHz band. To achieve this, the modulation method was changed from Direct Sequence Spread Spectrum (DSSS) to Orthogonal Frequency Division Multiplexing (ODFM). Further enhancements have been adopted for networks in the 2.4 GHz frequency band as well as in the 5 GHz band and most recently also in the 6 GHz band (which increases the data rate and transmission quality/efficiency in part with the aid of multi-antenna systems or channel bundling as well as other mechanisms).

Generational name	Technology supported
Wi-Fi 7	802.11be (in development)
Wi-Fi 6	802.11ax
Wi-Fi 5	802.11ac
Wi-Fi 4	802.11n

Figure 1: Generational naming scheme of the Wi-Fi Alliance and corresponding IEEE standard version

2.1.1 Functionality

Wireless LANs are networks in which the participants must share the transmission channel, a so-called shared medium. This means that only one station (STA) can transmit at a time if a collision of radio transmissions is to be avoided. A corresponding control mechanism is therefore required.

The IEEE 802.11 standard provides three fundamentally different architectures for transmission: The common case is the connection of several stations (STA) to a so-called access point (AP). This, known as infrastructure mode, enables the stations to be connected to other networks through the access point (the access point is usually equipped with several Ethernet-capable wired ports). If there is only one access point in such a wireless network it is called a Basic Service Set (BSS). An architecture that bundles several APs is called an Extended Service Set (ESS); this consists of several BSSs and a station can switch between the BSSs. A station connected to an ESS perceives the BSSs of an ESS as an overarching service set. In addition to the Infrastructure Mode the Ad-hoc Mode has been specified: This allows two stations to establish a radio link without an AP and thus exchange data directly (Independent Basic Service Set). The third mode is the mesh mode, which, similar to the ad hoc mode, does not require an AP, but can connect more than two stations to each other in a mesh BSS (MBSS). In order for a station to associate with a BSS it must know on which radio frequency communication with the AP can take place. This is made possible by the station iterating through the radio frequencies specified by the standard (the radio channels) and listening for special data frames, the beacon frames, prior to

association (scanning process). These data frames contain information that enables the station to establish a connection with the AP. Alternatively, a station can also actively ask for a BSS by means of probe requests.

2.1.2 Standard Extension 802.11ax (Wi-Fi 6, High-Efficiency Wi-Fi)

The 802.11ax standard is the successor to the 802.11ac standard (also Wi-Fi 5). The general conditions have not changed (same channel bandwidths and MIMO), only the 2.4 GHz band is now also addressed again. What is new, however, is modulation with OFDMA, for which the support by 802.11ax-compatbile stations must be given [5]. In theory, this allows higher network efficiency at high radio density (many subscribers on one channel). The standard also allows the use of the frequency range at 6 GHz. In addition, the target wake time (TWT) mechanism makes it possible to reduce power consumption for stations, since it is possible to coordinate centrally how often a device should wake up for data transfer. The utilization of the channel can thus be further optimized since stations do not use the channel unnecessarily. Furthermore, non-AP stations can now also use MU-MIMO (i.e. in the upload to the AP (UL)). This was previously only possible in the download. Now bidirectional MU-MIMO is possible. Target wake time should reduce the energy consumption of STAs and reduce the efficiency of the network through lower airtime. Also, Stations that support High-Efficiency (HE) have to support 802.11ac in 5 GHz or 802.11n in 2.4 GHz networks, too if they want to operate in that band [5].

2.2 6 GHz Wi-Fi Networks (Wi-Fi 6E)

The 802.11ax standard also specifies the use of the frequency band from above around 6 GHz. The use of this range is also called Wi-Fi 6E in the Wi-Fi Alliance generation scheme. The frequency range specified for this in the USA is 1200 MHz (5925-7125 MHz), a significant increase over the width previously permitted in the 5 GHz band. This therefore makes it possible for the first time to make sensible use of 160 MHz channels, which are usually out of the question at 5 GHz due to heavy utilization and regulatory restrictions. The spectrum is also not occupied by sources of interference such as weather radar, so there is no need to resort to DFS. During the realization of this work (February to May 2022), the spectrum band is also not expected to be used by other participants, so little or

no interference can be assumed. Of course, the other advantages and changes of the 802.11ax standard can also be used in the 6 GHz band: OFDMA, MU-MIMO, TWT and 1024-QAM as well as transmit beamforming are also possible here.

Figure 2: Spectrum and channel allocations for 6 GHz in Germany

2.2.1 Regulatory Context

The regulatory authority responsible for the EU, the ETSI, regulates the use of the 6 GHz frequency range for wireless access systems (WAS) in TR 103 524 [6]: According to this the frequency range between 5925 MHz and 6725 MHz is to be used for wireless access systems or radio local area networks (RLANs). There is no mention of a restriction on transmitting power compared with operation at 5GHz.

The Federal Network Agency has imposed further restrictions for Germany compared with the limits and frequencies previously permitted in the USA and compared with the ETSI recommendation: The general allocation from Order

55/2021 [7] specifies that use is permitted only from 5945 MHz to 6425 MHz. This extends the guard band at the lower end (towards the 5 GHz band) by a further 20 MHz compared with the ETSI recommendation. At the upper end even 300 MHz less are allocated. This allows the use of only three 160 MHz channels, since the other 160 MHz channels usable in the other U-NII bands in the USA are not available. In addition, only the use of Low Power Indoor (LPI) devices and Very Low Power (VLP) devices is permitted according to the following conditions:

Table 1: BNetzA regulatory limitations for LPI Devices

Low Power Indoor Devices	
Usable frequency range	5945 – 6425 MHz
Maximum EIRP for in-band broadcasts	0,2 W or 200 mW (23 dBm)
Maximum EIRP-density for in-band broadcasts	0,01 W/MHz or 10 mW/MHz
Maximum EIRP density for out-of-band emissions below 5935 MHz	$6,3 \times 10^{-6}$ W/MHz
Permissible operation	Limited indoor use, also in trains and aircraft. No outdoor use.

Table 2: BNetzA reguatory limitations for VLP Devices

Very Low Power Devices	
Usable frequency range	5945 – 6425 MHz
Maximum EIRP for in-band broadcasts	0,025 W or 25 mW (14 dBm)
Maximum EIRP-density for in-band broadcasts	0,00125 W/MHz or 1,25 mW/MHz

Very Low Power Devices	
Maximum EIRP density for out-of-band emissions below 5935 MHz	3,16 x 10^{-8} Watt/MHz
Permissible operation	Indoors and outdoors. Not for use on unmanned aerial vehicles (UAS).
Device category	The VLP device is a portable device.

This is a strategic disadvantage compared with the power of up to one Watt permitted in the 5 GHz band (5470-5725 MHz) [8], because the range of radio transmission for example at 6 GHz is much shorter with this limit, even indoors. For outdoor areas the reduction of the emission maximum to 25 mW (6 GHz) (in comparison: 1 Watt at 5 GHz) is extreme. Here, only close-range use is to be expected.

2.3 IoT and Device Management

The "Internet of Things" or IoT is seen as a collective term for the networking of objects or devices through communication technologies. In particular wireless communication comes into consideration here. The embedded computers thereby simplify or improve people's lives by increasing comfort or by adding new interaction possibilities or data. These physical devices, mostly sensors and/or actuators or combinations of these types to more complex structures (e.g., heat pumps or environmental sensors, smart appliances in the kitchen or industrial machines), are usually managed by a virtual representation on the Internet or an Internet-like structure [9].

The functions implemented on the Internet of Things allow interaction or management of the networked devices by humans or by automation. In detail, network devices such as access points or wireless clients (STAs) that provide or use wireless communication can also be seen and implemented as part of the Internet of Things. In the context of this work, therefore, the management of these

WLAN devices in the 6 GHz band is considered. Device management in the IoT environment includes the following points in particular:

1. Provisioning and deployment: Devices should be able to connect to the management software solution on their own and retrieve their own configuration from there.
2. Authentication: Devices must be centrally managed (device identity management) and have a secure channel for authentication.
3. Configuration: Devices and their internal parameters must be adjustable and automatically configurable from the management software solution.
4. Control: Commands or actions should be centrally triggerable on the active devices and the state of the device should be changeable (in the context of this work, e.g., start a throughput measurement on a Wi-Fi client).
5. Monitoring: System metrics should be centrally recordable and can be presented in an appealing way for the user.
6. Security: Devices mainly use a secure communication channel (e.g. HTTPs, mTLS or similar) for communication and actions in the software by the user are covered by a security concept (e.g. role-based access control).
7. Diagnostics: The device status is recorded and is visible to the user; Logging data and metadata can be accessed.
8. Up-to-dateness: The installed device software can be updated centrally as well as managed; The up-to-dateness of the device configurations can also be managed and viewed.

3 Technical Framework

For this work, primarily freely available open-source software was used where possible. This means that the measurement systems (server and client) are based on Debian GNU/Linux in the unstable "Sid" version. Debian stable at this time was Debian 11 "Bullseye". A few packages, such as the regulatory database (wireless-regdb), are only up-to-date enough in Sid to enable 6 GHz. A self-compiled kernel in version 5.17 also had to be used, as no official kernel build was available at the time that could correctly use 6 GHz with the provided hardware (Intel AX210).

Two slightly different systems were used for the hardware: A Dell Optiplex 9020 with Intel Cire i5-4570, 8GB RAM and 256 GB SSD was used as the first wireless client. Here, the Intel Wireless NIC is connected via an mPCIe adapter via PCIe. The server system (where the measuring station/server runs) uses an HP EliteDesk 800 G2 Mini with an Intel Core i5-6500, 8 GB RAM and a 256 GB SATA SSD. The system has a USB 3.1 Type-C interface, via which a 2.5 Gbit/s-capable Ethernet interface (chipset: RTL8156B [10]) is connected (for throughput measurements against an external server). The Intel Wireless NIC is connected here directly via the available M.2 slot.

The Intel AX210 Wireless NICs are not suitable for working as APs in a radio cell, so access points from the manufacturer Aruba are used. Magellan netzwerke GmbH [11] kindly provided two Aruba 630 series access points for use in this project.

The APs and server were connected via a Netgear GS110EMX multi-gigabit switch to enable wired communication.

3.1 Wi-Fi 6E

Establishing communication via Wi-Fi 6E is essential for the implementation of the project: Hardware is therefore needed that can use the 6 GHz band and that is available on the market[2]. For client NICs, only the following NICs from the manufacturer Intel were available:

a. Intel Wi-Fi 6E AX210 Gig+ M.2 Module
b. Intel Wi-Fi 6E AX211 Gig+ M.2 Module
c. Intel Wi-Fi 6E AX1675 Gig+ M.2 Module

In the following text option (a) will be abbreviated as AX210. Option (b) is identical in construction to the AX210 NIC, but with an Intel-proprietary interface communication via M.2, so it can only be used in hardware systems intended by Intel for this purpose and is therefore ruled out for this project. Option (c) is also identical in construction to the AX210 but does not have all the management options (no Intel vPro [12]) and was also recently about twice as expensive.

Aruba AP-635s are used for the access points. APs from Extreme Networks (AP 4000 series) were also requested, but these could not be provided by the manufacturer in time.

3.1.1 Intel Wireless NICs

Intel's AX210 enables tri-band 2x2 communication via WLAN as well as the use of Bluetooth 5.2. The NIC used is available as an M.2 2230 plug-in card. It supports gross data rates of up to 2.4 Gbit/s at 6 GHz and 160 MHz channel bandwidth, as well as MU-MIMO and OFDMA. [13].

a) Driver, Firmware and EEPROM

Due to the new general allocation of the Federal Network Agency for 6 GHz in Germany, a client cannot use this band without further ado as the network card itself may not yet be cleared for the frequency band. This is the case with the AX210: According to Intel, the card itself manages a list of permitted channels per location. If, for example, the current kernel 5.16 with the latest Intel driver and the latest Intel firmware is used for the card, it is still not possible to use the 6 GHz

[2] During the implementation period of this work, supply problems and bottlenecks occurred in the context of the Corona pandemic, which also affected chips and electronic products in particular.

band in Germany [14]. This is related to the internal determination of the regulatory domain in the Intel firmware:

1) Determination of the regulatory domain

Intel Wireless NICs use the Netlink interface under Linux to offer their hardware:

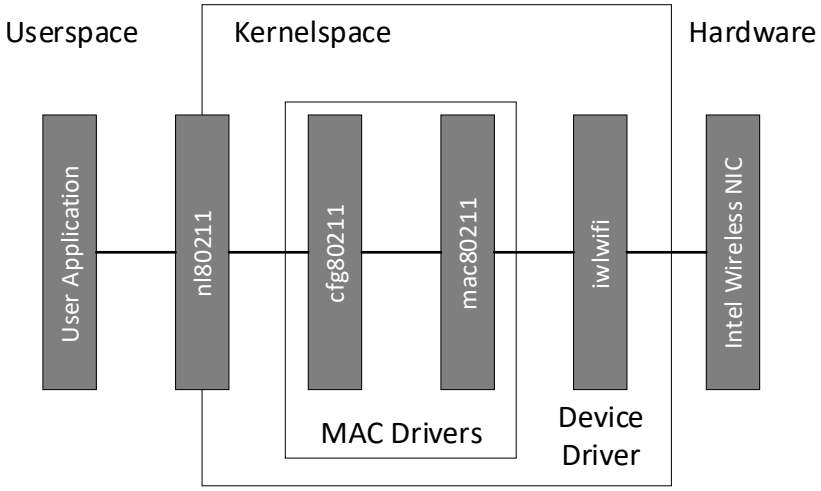

Figure 3: Access to an Intel Wireless NIC via the Netlink interface of the 802.11 driver stack (here with iwlwifi driver)

The operating system maintains a so-called regulatory database (for Linux in the package wireless-regdb), which contains the regulatory restrictions for each country, e.g. maximum transmission power, usable channels, DFS etc. [15]. This list can be accessed via the CRDA module, the Central Regulatory Domain Agent, in order to retrieve the restrictions on use for the hardware [1].

A wireless NIC registers with the Netlink interface via the cfg80211 MAC driver. In doing so, the NIC must provide an API that Netlink can use to specify the regulatory rules that apply to the device. The device therefore specifies during registration which frequency bands and which channels are supported therein. During registration, the cfg80211 driver checks against the regulatory database which rules apply to the current location and forms the intersection of the sets (i.e. only those operation modes remain that are both supported by the device and allowed by the regulations).

In addition, the firmware of the Intel NIC itself can now further tighten the rules. With Intel, this is done by the "Location Aware Regulatory" (LAR). The exact mode of operation is not published by Intel, but it can be assumed that the card itself

performs a scan and determines, on the basis of the received management frames, in which state/regulatory domain it is located. Then the card uses a list stored in the EEPROM and managed via the firmware, similar to the Regulatory Database, to restrict the usable frequencies and transmitting powers etc. This means that even if, for example, US is set as the regulatory domain via the CRDA, as long as the card is located in Germany (DE) and receives at least one management packet with a different country code, the card will continue to restrict itself. Therefore, although the 6 GHz band can be used in the USA, the card cannot use this band.

In the test phase of Intel's LAR it was possible to switch off this functionality via a driver option (`lar_disable`). This is no longer possible [16].

2) Solving the regulatory domain problem

However, since 6 GHz can actually also be used in Germany since 2021, but this was not possible at first, Intel was informed accordingly [14]. The solution is to use the Linux 5.17 kernel, which presently was not yet available as a compiled package for Debian during this work, and to use a specific firmware version from Intel. With this firmware version, the Intel AX210 selects the correct list of frequencies and allows operation as a station in the 6 GHz band. Only the display and retrieval of some parameters (e.g. transmission power), as well as operation as an Access Point, do not yet seem to function completely correctly [14].

3.1.2 Aruba Access Points

The access points were provided to us by Magellan Netzwerke GmbH, Cologne. They are AP-635 (model with internal antennas for indoor use) from the Aruba 630 Wi-Fi 6E AP series. [17]. Aruba is a Hewlett Packard (HP) company. The Aruba 630 series allows simultaneous use of all three bands (2.4 GHz, 5 GHz and 6 GHz) and has two 2.5 Gbps Ethernet ports that can be used as uplinks. The access point supports 2x2 MIMO on all three bands [18]. This means that at 6 GHz and a channel bandwidth of 160 MHz, up to 2.4 Gbit/s is theoretically possible as a gross bit rate.

According to the manufacturer OFDMA, TWT, transmit (TX) beamforming and BSS coloring are also supported. In the 5 GHz band, allocating up to 8 OFDMA Resource Units (RU) are supported, in the 6 GHz band even up to 37 RUs. The

maximum transmit power is 21 dBm [18] (without antenna gain). The AP-635 is designed for ceiling mounting. The AP can be powered via PoE+ (802.3at) directly through a connected Ethernet cable.

3.2 IoT Management Software

In the course of initial research the following (open-source) candidates for the software-side management of IoT devices were elaborated:

1. OpenRemote [19]
2. Thinger.io [20]
3. ThingsBoard [21]
4. Mainflux [22]

For these four software options, the points listed in chapter 2.3 were then used to work out the extent to which the software solutions basically meet the requirements. For this purpose, the documentation of the respective software solutions was used. A points-based evaluation scheme was worked out, the complete table for which can be found in the appendix. The two solutions that achieved the highest and second highest scores were selected for further comparison of function and evaluation with regard to Wi-Fi: ThingsBoard and Thinger.io.

The other software solutions were discarded for further consideration, as they did not meet some requirements: OpenRemote, for example, does not allow the configuration of the device to be backed up (MUST requirement), Mainflux does not inherently allow any control options for the managed devices, and also does not provide a front-end through which a user with further in-depth technical experience can operate the IoT devices.

3.2.1 ThingsBoard

ThingsBoard is an open-source [23] IoT platform for device management, data-collection as well as data processing, which also prepares the data graphically in a frontend. Industry standards such as MQTT and HTTPS are used to connect to the devices. Both local installations and installations in the cloud are possible. ThingsBoard offers server-side APIs for the overall management of the devices through which the IoT platform itself and the devices can be managed, controlled, and monitored. The platform allows multiple customers/tenants to be managed in their own separate environment and devices/assets can be assigned to customers respectively. Telemetry data can also be collected centrally via the API and can be prepared in dashboards. The dashboards can then also be used directly by the

customers. For the telemetry data, so-called rule chains can be used for data processing. This allows the data to be processed and transformed. Alarms can be triggered by rule chains, attributes of the devices can be updated and actions can be initiated.

ThingsBoard offers online documentation [24] and was released in 2016 in the first major version 1.0. The most recently released version is 3.3.4. The software can be used free of charge in the Community Edition. A paid Professional Edition is also distributed, which offers additional support options and customization of the software to a corporate design.

3.2.2 Thinger.io

Thinger.io describes itself as an "Open Source Platform for the Internet of Things" [20]. The focus is on connecting and managing IoT products. The software integrates with the devices via its own client software or via a REST API. The software can also be installed on-premise as a container stack, or, alternatively, Thinger.io's own cloud solution can be used. Both options incur costs, ThingsBoard is only free with up to a maximum of two managed devices. Especially the Arduino Ecosystem is in focus: These devices (such as ESP8266 or Arduino MKR 1010) are directly supported by the Thinger.io library. However, MQTT- and HTTP-based devices can also be connected. The software supports the management of devices from the frontend, collection of data in so-called "data buckets" [25] and the visualization of data in dashboards. The software has been under development since 2015. In the context of this work, version 3.4.6 was evaluated in a local deployment in the "Medium" license plan, which was made available by the manufacturer upon request.

4 System Design

The required functionality, i.e. the management of the Wi-Fi devices (like IoT devices) as well as the measurements are mapped by a client-server architecture. At least necessary is a client-side measurement program, which is executed on one of the client PCs described above and the server, which takes over the management (on which the ThingsBoard software or Thinger.io software is executed). In addition, it is useful to set up an independent server that can collect the measurement data and metrics independently from the management software, so that in case the desired result cannot be achieved with the software, at least Wi-Fi 6E can be evaluated without any problems caused by shortcomings from the management software. In addition, one of the (measurement) clients should also be able to act as a measurement counterpart, i.e. in this sense as a measurement server for the performance measurement. The client program itself should be able to communicate with the management software in encrypted form, using at least the standard HTTPS protocol; in the event that the measurement data is written to an independent system, this should also be done via HTTPS for the sake of clarity.

At the physical hardware level it is important to ensure that the measurement path, i.e. all components between the measurement client and the measurement server (e.g. Ethernet interfaces and switches), also support at least the theoretical maximum data rate of the wireless connection. Otherwise, it cannot be ensured that a restriction of the measurable performance does not occur there, which is not due to the actual wireless connection but created by the limiting wired link.

4.1 Architectural Design

The setup contains the following components:

1. ThingsBoard-Server
2. Thinger.io-Server
3. Netgear-Switch
4. 2 × Aruba-Access-Points
5. HP-Client (Used partially as a measurement server)
6. Dell-Client

The majority of the components can be clearly differentiated from each other: ThingsBoard and Thinger.io servers each serve to centrally manage the clients used: They are to provision, control and configure the devices. Likewise, they are to receive data such as telemetry, measurement data, etc. from the clients. The Netgear switch is capable of connecting the components at 2.5 Gbit/s via Ethernet, but it does not support PoE+, which the Aruba APs need in order to operate. A PoE injector, which is also 2.5 Gbit/s-capable, is provided for this purpose.

The Aruba access points were placed in the lab with the help of a spatial survey so that testing the roaming ability of clients can be carried out at an easily accessible point in the central corridor. The transmission power of the APs was also reduced accordingly for the time being in order to be able to define the roaming point well (This was later adjusted for maximum throughput). This should simplify the measurement of the roaming behavior of the clients.

The HP client is equipped with a 2.5 Gbit/s Ethernet adapter via USB 3.1 Type-C in order to be able to be used not only as a Wi-Fi-based test station (e.g., for tests between clients in the same cell) but also as a wired test station behind an access point via the switch. This enables the measurement of a single client in a cell. The Dell client is intended as a measurement client and is to run either the client program for ThingsBoard/Thinger.io or the independent measurement program.

4.1.1 Network Plan

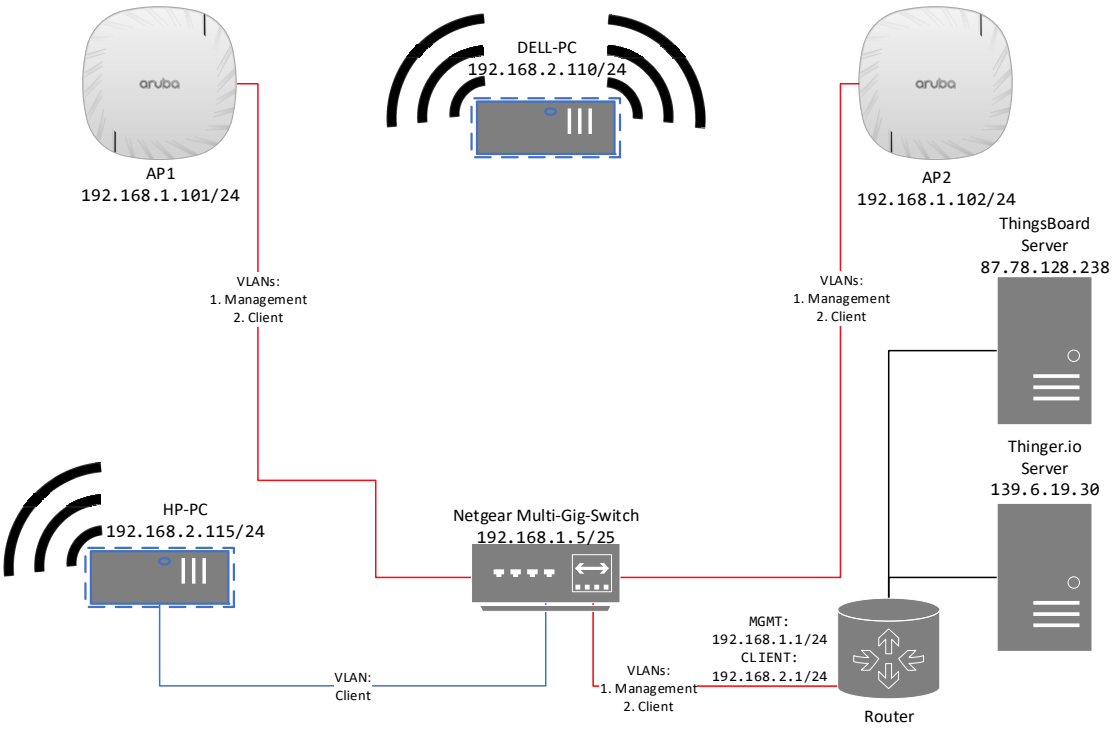

Figure 4: Network plan/setup of the network for the test environment in the laboratory. The HP PC either takes the role of the server (via cable or wireless) or is used as a second client. The images of the Aruba access points are designs of the VSD Grafx Inc [26].

The access points are connected to the switch via multi-gigabit-capable ports, so that the total theoretically possible data rate is not restricted by the Ethernet connection during cross-cell communication. The Dell PC, since it is operated exclusively as a client, does not require a wired connection, but only has a radio connection. For this wireless connection the data is collected and it is over this connection that the QoS and performance measurements are made. For the scenario of communicating with a second wireless device, the HP PC can do without its wired connection, since it is also equipped with an AX210 interface and therefore also supports Wi-Fi 6E.

Not drawn in the diagram above are the PoE+ injectors used between the switch and the APs. They are not relevant for the logical structure since they only ensure the power supply of the APs.

Internally two private IPv4 networks are used to separate the management and client networks. The 192.168.1.0/24 network can be used to manage the router, the switch, and the APs via a virtual controller running under the IP 192.168.1.10 (not shown in the figure above, as it can move between the two APs). The two clients talk over a network logically separated by a VLAN: 192.168.2.0/24. The servers for the ThingsBoard and Thinger.io applications are hosted externally and are accessible via a router, which simultaneously enables DHCP in the respective network as well as NAT for the APs and the clients.

4.1.2 Software Design: ThingsBoard

ThingsBoard manages so-called entities. These can be, for example, a tenant, a customer, a user, a dashboard or a device [27]. Devices can send telemetry data to ThingsBoard and respond to RPC commands. These can be sensors or actuators, to name some examples. In this instance a device is a Wi-Fi 6E-enabled Linux computer that can also collect telemetry data (e.g., channel quality of the connection) and should respond to commands (e.g., start a performance measurement).

A device not only collects telemetry data, but also has self-defining properties. ThingsBoard names these as attributes. These are key-value pairs that belong to the device. This is used in the context of this work, for example, to store and change the configuration of the device centrally or to log the firmware version of the Intel firmware.

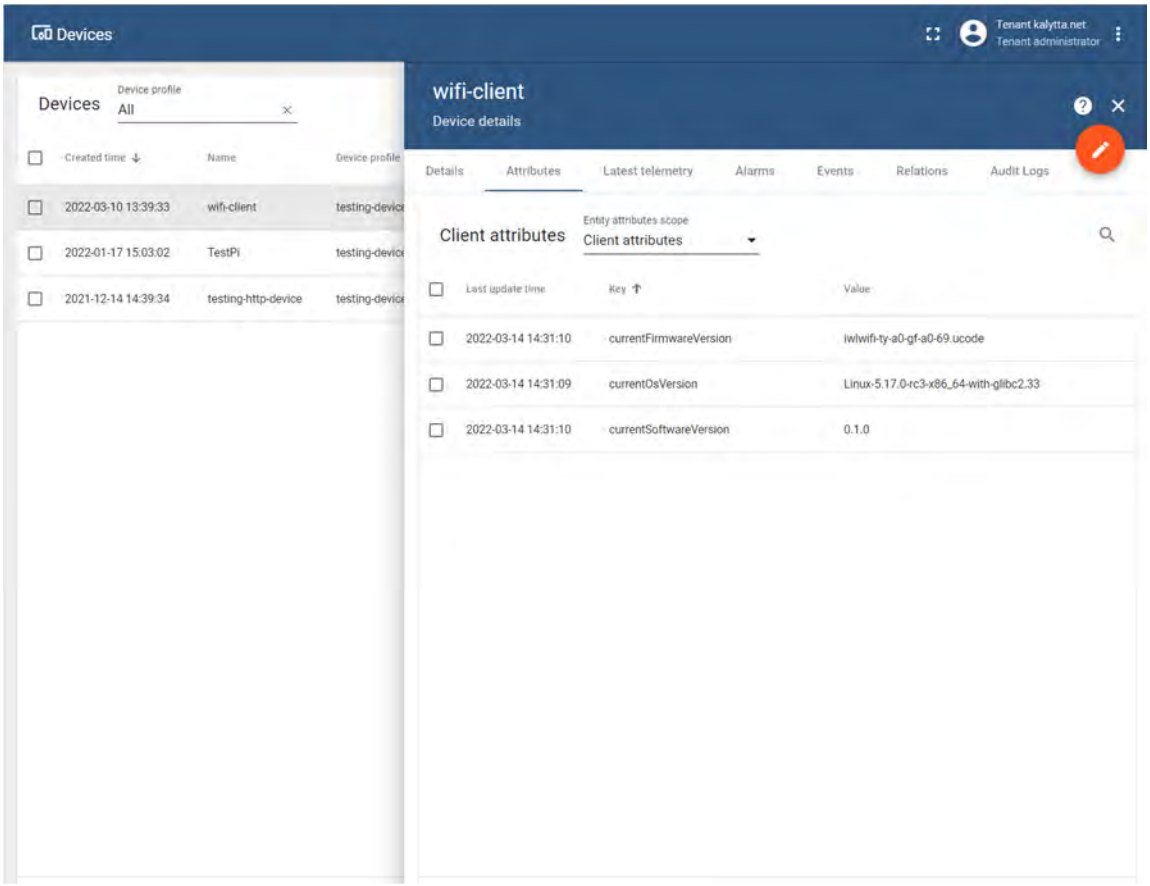

Figure 5: ThingsBoard device details show for example client attributes that can contain information like firmware version or operating system information

Devices can additionally send time series data to ThingsBoard: Either directly as telemetry (this is stored in a central database as a JSON object) or as a response to an RPC call (also JSON-based). This can then be used in dashboards to prepare, for instance, the measurement data sent in this way. This data can also be processed in a rules engine or react to unusual data points (alarms).

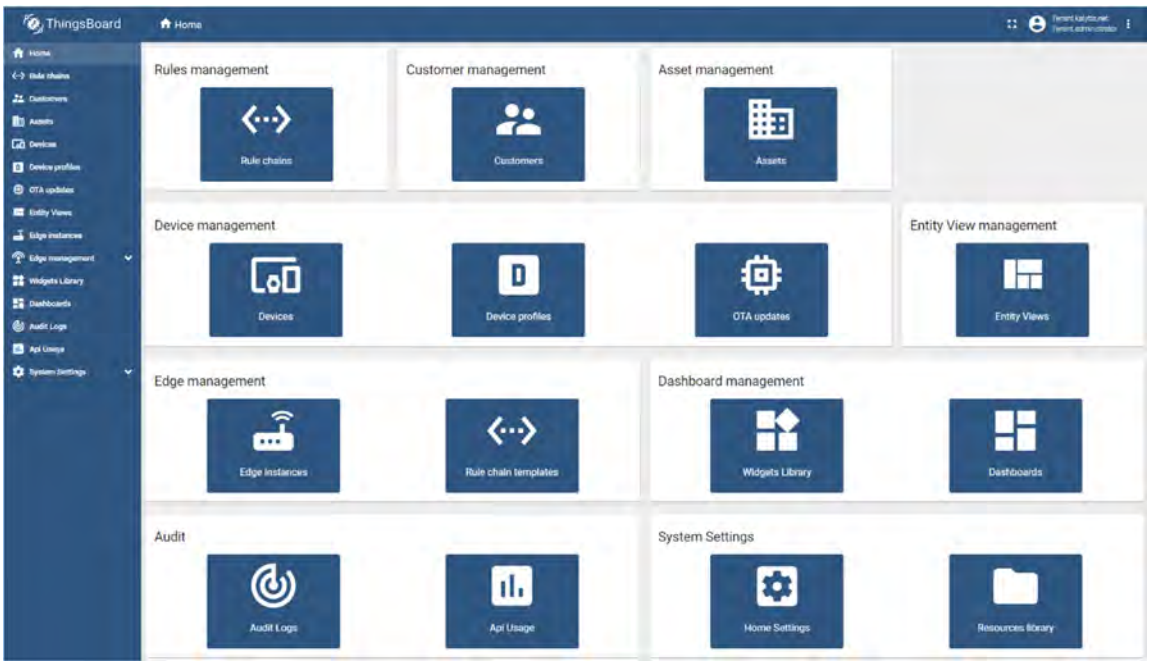

Figure 6: ThingsBoard web overview: The different entity types are visible as well as the more specific points as over-the-air updates and the dashboard management

ThingsBoard offers various possibilities to connect client devices: In addition to an MQTT API, CoAP, LWM2M, HTTP and SNMP are available. The Linux client, as a powerful computer, can easily make complex API calls and therefore also use a stateful protocol like HTTP. HTTP is also supported by Thinger.io, so it was chosen as the protocol for communication between the platform and the device in both cases. This makes it possible to potentially reuse some of the communication logic.

4.1.3 Software Design: Thinger.io

According to the manufacturer, Thinger.io allows bidirectional communication of the server with any client devices, regardless of the hardware platform. The devices can be assigned to a client.

Devices can be connected via MQTT, Sigfox or via LoRaWAN. Alternatively, an HTTP (RESTful) API can be used to interact with Thinger.io. Through the endpoints a device can receive and send JSON data. Communication is possible over encrypted HTTPS and the client must authenticate via an Authorization header. Sent JSON objects are stored in so-called data buckets. Each device has properties that are comparable to the attributes in ThingsBoard. These are JSON objects that are uniquely assigned to a device and can, for example, contain the configuration.

Here, however, a limitation of Thinger.io already becomes apparent: Since the Wi-Fi 6E clients are much more complex than a simple IoT sensor or actuator, not only data of one type must be stored, but different types (configuration, telemetry, measurement data, system state, etc.). However, a Data Bucket can only contain one type of data at a time. If a client now writes configuration data to its assigned bucket, the client will only be able to write configuration data to the bucket and read it from there. If it writes other data to it, the bucket can no longer be used on a dashboard to display data, because it is not clear which of the various data sets should now be displayed. Also, only one property can be sent to the device in response to a request from the device. This is problematic since a device may send either telemetry data or properties. I.e. a device that sends measurement data cannot update its own configuration (e.g., when it switches between two WLAN service sets) and a device that should be able to change its configuration cannot send measurement data. This makes the Thinger.io approach unsuitable for use with complex devices that combine multiple tasks in one endpoint.

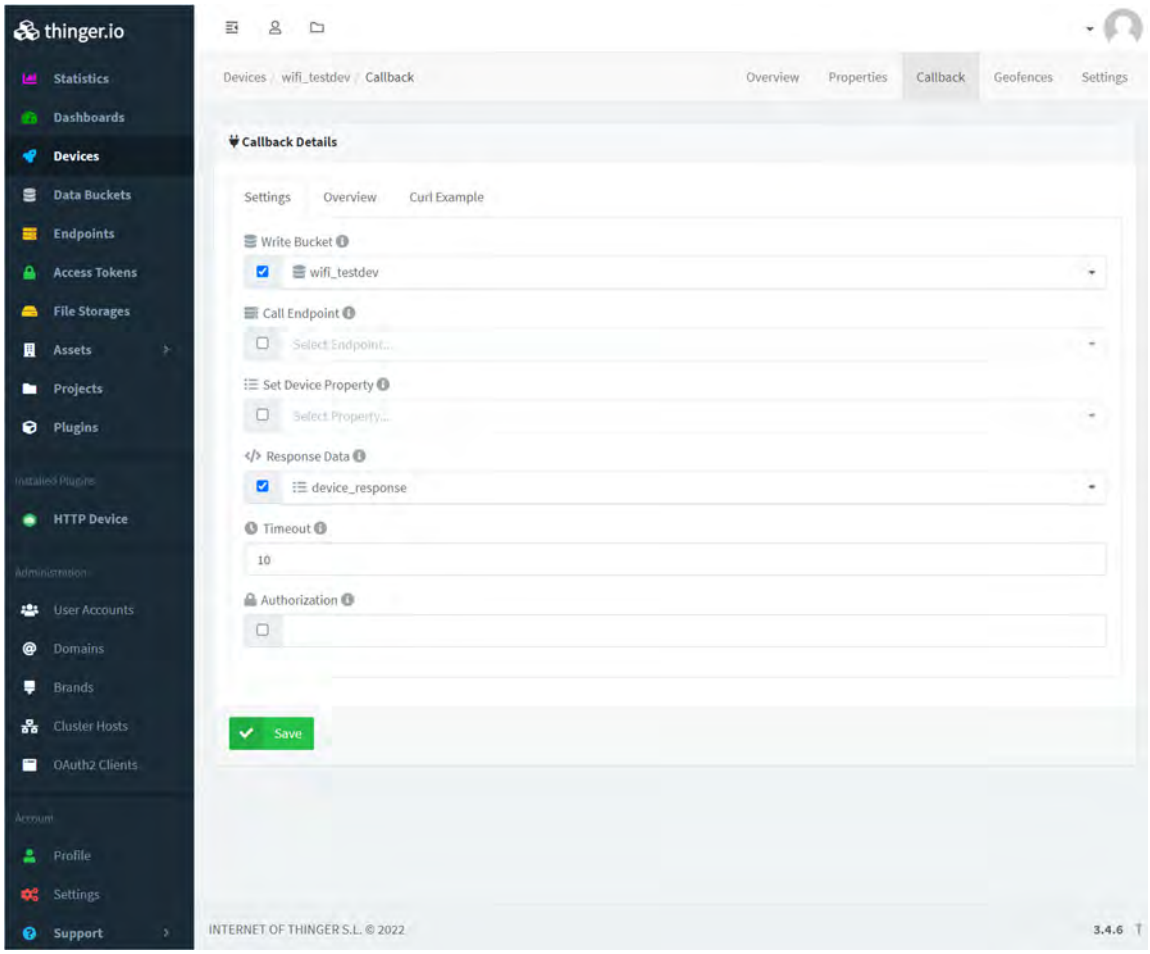

Figure 7: Thinger.io device configuration allows only one data bucket to write to. Also, only one device property can be sent to the client.

Additionally, it should be mentioned that the connection of HTTP devices is directly presented as a feature in Thinger.io by the vendor [28], but is actually implemented via a plugin that is not further documented: The documentation of the plugin was removed during the writing of this paper, but was originally available during the evaluation phase [29]. Automatic provisioning of devices can also be performed via this plugin.

Since Thinger.io already does not fulfill basic requirements that are relevant for the objective of the work, a direct comparison of the software with ThingsBoard regarding the performance in the management of Wi-Fi 6E devices cannot be carried out. Therefore, only a functional comparison of the two software solutions will be carried out in the further course of this work. Only the features of ThingsBoard are then considered in more detail with regard to the management of Wi-Fi 6E devices and a client program is implemented that allows the management by ThingsBoard.

4.2 Wi-Fi 6E Measurement Environment and Scenarios

In order to be able to perform an assessment of the current status of 6 GHz Wi-Fi networks, the parameters that are relevant from the user's perspective in particular must be considered, i.e. also the parameters that represent the innovations from a technical perspective, which can then be evaluated in more detail in test measurements. From the user's point of view, the following parameters were considered relevant, with a particular focus here on the first four parameters, which are usually used as metrics determining the quality of service (QoS):

a. The data throughput for applications (TCP/UDP)
b. The latency between client and server (round-trip time)
c. The jitter of the latency
d. The availability of the data connection for longer lasting transmissions
e. Roaming Behavior of Client Devices in an Extended Service Set
f. Behavior of the clients with increasing distance to the next access point (distance limits)

Other scenarios that can occur in a multi-client environment can also be considered (such as hidden nodes) but these are not considered further in this paper.

From a technical point of view, it is also interesting to consider the newly defined MCS for High Efficiency WLAN (indices 10 and 11) and the possibility of communicating with OFDMA as a modulation in comparison with the OFDM modulation that has been common up to now.

4.2.1 Quality-of-Service

A subscriber in a WLAN network expects a reliable connection to the desired destination. The quantitative recording of the above-mentioned parameters is interesting here: Latency, jitter, throughput and availability. It is therefore necessary for the measurement system to be able to record the latency, jitter and throughput of the wireless data connection at regular intervals. Longer-term measurements of throughput should also be possible, so that fluctuations in availability or lost data packets can be recorded. If these measurements are performed for a 5 GHz 802.11ac-based network and a 6 GHz 802.11ax-based network, for example, the parameters can be compared on this basis and statements can be made about differences or similarities. In order to optimize the throughput the measurements are performed with the maximum possible channel bandwidth of 160 MHz (6 GHz band) and 80 MHz (5 GHz band).

4.2.2 Roaming

The Federal Network Agency has explicitly earmarked the 6 GHz band for mobile devices as well [7], e.g., smartphones, tablets, mobile IoT devices, robots or comparable mobile hardware. In extended service sets (more than one AP), it is therefore inevitable to expect devices to be roaming. This is also an integral part for users to achieve good quality of service. If a client moves too far away from the associated AP, standalone (802.11k), or AP-supported (802.11v) roaming should be enabled if other APs are available. Of particular interest here are threshold values (received power, link quality) at which the clients switch, and how high the stickiness (delay in switching between two APs or clients that do not switch at all, although the connection is getting worse) is. Here, the selection of parameters such as the transmission power of the adjacent access points

between which the handover is to take place, which are decisive for roaming, is then also interesting.

4.2.3 Distance Measurements

IoT devices in particular can be distributed over the entire site or building and therefore also have large distances to the next access point. For this reason, the measurement of the performance for increasing distances is also important with regard to the low maximum transmission power in the 6 GHz band. In addition, the increased frequency also means that there is possibly already a measurable difference in the RF reception parameters compared with 5 GHz networks. The measurement of the QoS metrics listed above is therefore also performed with increasing distance to the AP. Since the lowest possible complexity is required in this scenario, 20 MHz is set as the channel bandwidth.

4.2.4 MCS and Spatial Streams

It is also interesting to look at the newly defined MCS indices 10 and 11 for HE, which offer 1024-QAM and a coding rate of 3/4 and 5/6 respectively. This in combination with the lowest guard interval (0.8 µs) allows the transmission of more than one Gigabit (MCS-HE 10: 1080.9 Mbit/s, MCS-HE 11: 1201 Mbit/s) with one Spatial Stream (SS) at 160 MHz channel bandwidth. With 2 spatial streams even 2402 Mbit/s gross are theoretically possible. The measurements are to determine whether these MCSs can be reliably selected by the clients and what data throughputs can thus be achieved at the transport layer.

The MCS as well as the number of spatial streams is selected by the client and can also be retrieved there. The measurements are recorded in a background process so that these parameters, which are significant for the connection, are also recorded during a series of measurements. In the case of a distance measurement, for example, the distance or signal reception strength at which the client switches to a lower MCS can be recorded.

4.2.5 OFDM(A)

For the first time the 802.11ax standard allows the distribution of the individual carrier frequencies to several simultaneous participants within an OFDM symbol in both directions. This makes it possible to achieve true simultaneity of radio transmissions in upload and download. The standard only allows data frames to be transmitted via OFDMA; management and control frames continue to be transmitted via OFDM [30]. An access point must also use trigger frames to assign subcarriers to the clients that it is to use for OFDMA. The client must confirm this (via clear-to-send/CTS response). This behavior, if it occurs, as well as the actual OFDMA-based data transmission is to be observed and evaluated in the measurement series.

5 Implementation

The functionality to perform throughput measurements via WLAN was implemented on the program iperf3 [31], a throughput measurement tool that offers a wide range of configuration options. The program works with a client-server architecture. With iperf3, for example, the throughput on an interface can be recorded simultaneously over several parallel streams for a certain period of time by the client measuring the throughput to the server (or vice versa). The data can then be programmatically processed as JSON. Around this program as a basis, a Python 3 wrapper program has been created as part of this thesis, which has been extended with additional functions and wrapping program parts. These functions/program parts include (not exclusively):

- The use of iw as a program for controlling and configuring wireless interfaces (e.g. the Intel NICs used) as well as retrieving the interface parameters (MCS, channel bandwidth, spatial streams, etc.).
- Direct InfluxDB integration (via Python package) for storing captured time series data.
- Integration of the ping3-Python package [32] for ICMP ping to capture latency.
- The Python multiprocessing package for mapping the concurrent program parts (e.g. acquisition of channel parameters during a measurement series).
- The Python-ConfigParser package for reading and creating configuration files.
- The ThingsBoard REST Client for Python [33], for partial connection to ThingsBoard.

Python was chosen as the programming language because programs are easily portable and can be run on most client systems without problems. Integration with the IoT management software can be easily done via Python's HTTP package (such as urllib) and operating system modules or calls can be integrated directly.

5.1 Program Structure: ThingsBoard

For ThingsBoard the entire test functionality was implemented within one Python file. This makes a function update very easy by replacing the file with a new version. For storing the persistent configuration options an INI file for the configuration, client.conf, and a file for storing secrets is used.

The client software takes care of reading the local configuration, registering with the management platform, and all communication during program execution.

5.1.1 client.py

The client program contains various functions and routines that can be triggered by different mechanisms on the client or by the remote management software:

a) main()-Function:

This is the entry point when starting the client: first the configuration and secrets are read from the configuration files and a new instance of MeasurementClient is created, a Python class that contains the further functionality. Then the global logging instance is created, which can be used to keep a debug log. After that, the network connection desired in the configuration is established via WLAN.

```
def register_device(self):
    url = "https://" + REMOTESERVER + \
        ":" + REMOTEPORT + "/api/v1/provision"
    body = {
        "deviceName": DEVICENAME,
        "provisionDeviceKey": DEVICEKEY,
        "provisionDeviceSecret": DEVICESECRET
    }
    json_body = json.dumps(body)
    ca_path = self.config["CONNECTION"]["TrustedCADirectory"]
    response = requests.post(url, json_body, verify=ca_path)
    decoded_response = response.json()
    received_token = decoded_response.get("credentialsValue")
    if (received_token is not None):
        self.secrets["SERVER"]["Token"] = received_token
        with open(".secrets", "w") as secretsfile:
            self.secrets.write(secretsfile)
```

Code 1: register_device() function that allows a device to self-register it with the remote ThingsBoard server and obtain an API token for further communication

If there is already a registration with a ThingsBoard server in the Secrets file, then the main loop of the MeasurementClient is started, if not, then the client is registered by the `register_device()` function.

b) Main-Loop run_loop():

The main loop executes an endless loop over which the following functions are mapped and executed accordingly when the prerequisites are met:

1. Collect device parameters (attributes) and send them to the server (operating system version, client software version, firmware version for the Intel WLAN NIC).
2. check if new firmware versions are available on the server. If so, then install the latest version.
3. Wait for remote procedure calls (RPC) from the server.
4. If there is an RPC for a throughput measurement by the server, then the throughput measurement is performed via `do_throughput_measurement()` and the obtained measurement data is sent back to the server as telemetry data.
5. If there is an RPC for a latency measurement by the server, then the latency measurement is performed via `do_rtt_measurement()` and the obtained measurement data is sent back to the server as telemetry data.
6. Finally, additional system metrics are collected before the next loop pass and sent as telemetry (`send_telemetry()`): CPU usage, RAM usage, hard disk usage and data about the used WLAN interface (e.g. transmit power, ESSID, MAC address of the access point).

Sending telemetry data is basically possible via a dedicated API endpoint of the ThingsBoard server, which can be used with the authentication token obtained by registering the device: `/api/v1/<token>/attributes` allows the sending of device attributes as telemetry. However, it is not possible to set shared attributes (e.g., the client configuration) via token only. For this purpose conventional access data (user name/password) must be used.

c) update_firmware():

This function contains the exemplary handling of a firmware update where the updates are distributed centrally by ThingsBoard.

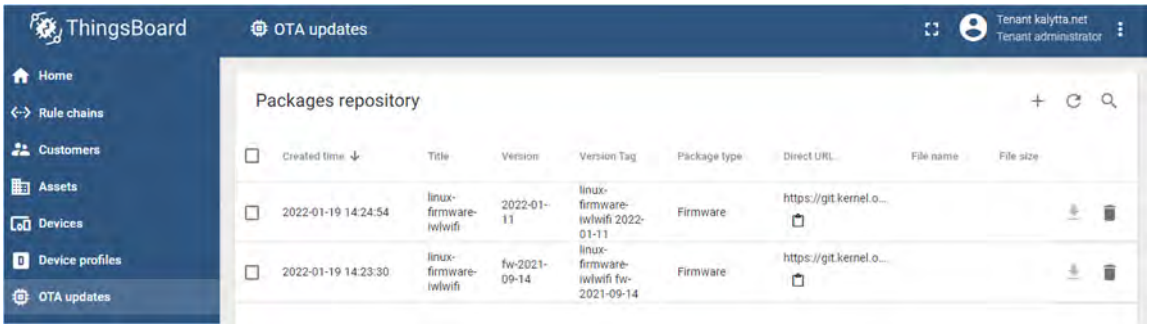

Figure 8: ThingsBoard Over-the-Air dashboard allows for upload or URL reference to a firmware or software file (package type) that can be pushed to devices or device groups (profiles) automatically.

After the client has verified that a newer firmware version is available the client retrieves the URL of the download file from the server and then performs the download. Firmwares are a tar archive compressed with gzip, which is then unpacked over the previously installed firmware. Afterwards the downloaded archive can be removed again. Now the local configuration file is updated (writing the new firmware version into the configuration file), so that the new firmware is not retrieved again.

d) wait_rpc():

The response of the client to remote procedure calls is mapped via the `wait_rpc()` function: Here, the client retrieves the first available RPC from the server via GET request and unpacks its payload. The payload contains an ID to identify the RPC (This could also be used by a client to send back asynchronous responses to an RPC) as well as the "method" field, which is filled with "rpcCommand" (Indicates that it is an RPC command). Furthermore, it can contain several parameters in the "params" field, which may contain, for example, various commands or their parameters:

```
{
    "id": 162,
    "method" : "rpcCommand",
    "params": {
        "command": "doPerfMeasurement"
    }
}
```

Code 2: Example RPC payload from ThingsBoard when the client receives an RPC command, in this case: doPerfMeasurement, which starts a 13 second iperf3 measurement on the client

In the case of a throughput measurement, this is carried out when the corresponding command is coded in the payload. Afterwards the client sends the measurement results as telemetry to ThingsBoard. Further commands are possible, e.g., doRttMeasurement, which calls the corresponding function for the latency measurement (see below).

e) send_telemetry():

The client uses this to regularly record system metrics such as CPU and memory usage as well as swap and local hard disk utilization. Information about the WLAN interface is also collected. These data are sent as telemetry data to ThingsBoard.

f) do_throughput_measurement():

The client performs a throughput measurement with iperf3 after disabling all other unused network interfaces. This ensures that, for clients which may still have an Ethernet interface or a second Wi-Fi interface, it is not preferred for the measurement due to routing. The exact execution of the measurement and the parameters used with iperf3 is described in chapter 5.2.1.

g) do_rtt_measurement():

This function enables the latency measurement to the Iperf3 server. Three ICMP echo messages are sent and the ping delay is returned in each case.

5.1.2 client.conf

The configuration of the client is stored in a file and contains the defaults for the initialization of the program as well as some changeable options: Generally, over a timestamp in the configuration file the last conditions are compared with the version of the client configuration held on the server. Thus, updates can be played out from server side to the client as well as the client can synchronize

configuration changes to the server. The newest timestamp is seen as source-of-truth.

One can also configure settings for connection to the ThingsBoard server and iperf3 server, as well as logging and a few parameters for the Wi-Fi interface (Used interface, country, transmit power and WPA configuration):

```
[GENERAL]
timestamp = 2022-03-11T13:39:34.899415
firmwareversion = 2022-01-11
[CONNECTION]
remoteserver = thingsboard.home.kalytta.net
#remoteport = 8080
remoteport = 443
devicename = wifi-client
trustedcadirectory = certs
[IPERF]
iperfserver = 192.168.2.144
iperfport = 5201
[LOGGING]
logfile = ./client.log
loglevel = DEBUG
[WIFI]
wlaninterface = wlp1s0
country = DE
txpower = 20
wpa_configfile = /etc/wpa_supplicant.conf
wpa_configfile_local = wpa_supplicant.conf
```

Code 3: client.conf configuration file allows for basic configuration of the client program, i.e. setting the remote server address for ThingsBoard

5.1.3 .secrets-File

Part of the configuration should be readable and writable only for the client program - this part is outsourced to a second configuration file (.secrets), which is also not synchronized centrally with the server. At the beginning on a new client, it contains the key and the secret for the registration as well as Wi-Fi access data for the provisioning with the server. The client token needed for API access is then added later. Once the file is updated accordingly it can be easily reused by the client program even after updates or reboots.

5.1.4 Server-side Processing

The server can respond to incoming telemetry through rule chains instead of just storing it (which is the default behavior). It can also respond to attribute changes or RPC commands from the client. Arbitrarily complex sequences of rules in a kind of tree structure are possible. In the case of an API call the root rule chain is always triggered. Depending on the message type this can then trigger various other rule chains or actions: e.g. messages can be filtered, data can be subsequently enriched or transformed (adding metadata, changing the data content based on a script) or alarms can be triggered:

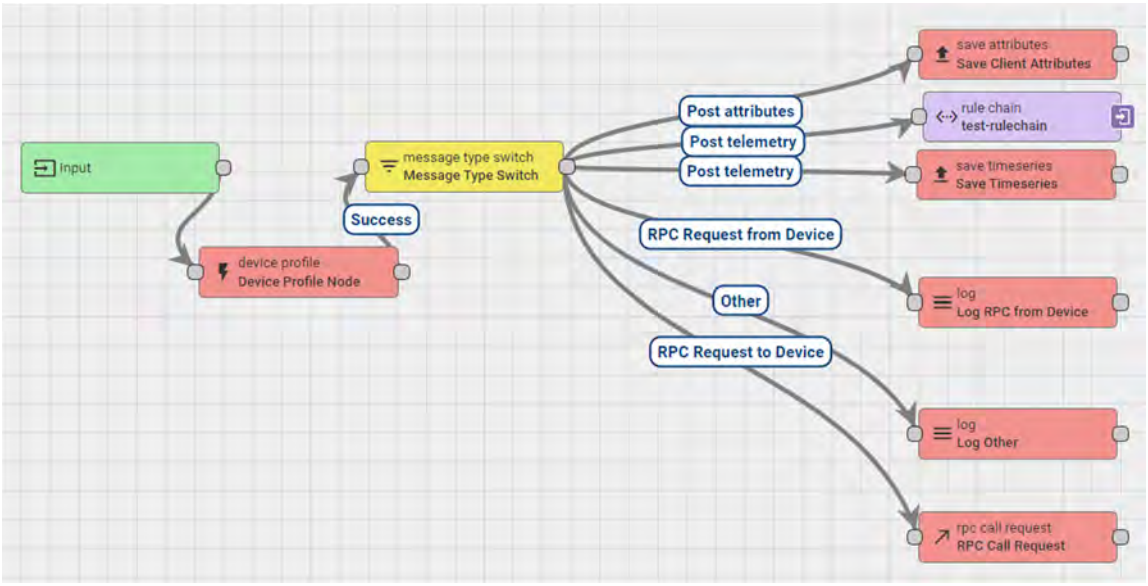

Figure 9: ThingsBoard Root Rule Chain: Allows for granular actions on API events: Here "Post telemetry" also calls another rule chain in a chained call.

The client can therefore also actively trigger alarms by, for example, packaging information in an RPC and a rule chain extracts and evaluates this information (e.g. via threshold values). Alternatively, it can also react to unusual changes in the configuration of a client (client suddenly changes the country).

5.2 Program Structure: Independent Measurement Client

The independent measurement client allows the measurement of throughput and channel parameters independent of a management software like ThingsBoard. The data generated by iperf3 is not sent here as telemetry or RPC response but stored locally as JSON object. In addition, this data is written to an Influx database as time series data to enable subsequent evaluation.

The independent measurement client is divided into two parts: The first part (iperf.py) serves as program entry point and performs the actual measurement as foreground process. In addition, this part is responsible for sending the measurement data to Influx. The second part takes care of measuring the channel parameters in the background of the actual throughput measurement and runs as a second process separate from the main part. This allows independent detection of variations in environmental variables (such as the strength of the received signal from the access point) during the measurement.

5.2.1 iperf.py

The iperf.py program controls the actual measurement process and makes the preparations for the measurement as well as performs the post-processing after the measurement. This file can be called directly as a command line program. The following parameters are supported:

Table 3: Parameter set of the iperf.py program

Parameter	**Description**
--protocol, -p	Switch for changing between TCP and UDP for the throughput measurement. Default is udp.
--interface, -i	Sets the interface that will be used for the measurement (This prevents the program from disabling it while measuring).

Parameter	Description
--duration, -d	Sets the measurement duration in seconds. Note that the actual measurement time will be double this time as upload and download are both measured this same amount of time. The first three seconds of data for each direction are omitted by iperf3 but are contained when using JSON output. Default is 60 seconds.
--bandwidth, -b	Target bandwidth for UDP Measurements. The sender will generate this amount of packets (i.e. setting this to 1G, the client will generate 1 Gbit/s of continuous UDP data). Default is 5 Gbit/s.
--streams, -s	Number of transmission streams for TCP and UDP. Default is 10 streams.
--windowsize, -w	Sets the initial TCP window size. Default is 512 Megabytes.

When called, the program first creates a background process to capture the round-trip times (via ICMP echo) between server and client and starts another process to monitor the interface and channel parameters (call interface.py). Then the throughput measurement is performed with the specified parameters. Here, it is to be noted that on the target server, which is to be defined in the program configuration, an instance of the iperf3 server must be started on port 5201.

First the client measures the download (i.e. the RX channel of the client is tested), directly after that the upload (TX channel of the client):

```
print("Start Measuring Download to Client")
try:
    result = subprocess.run(
            ['/usr/bin/iperf3', '-c', str(IPERF_SERVER), "-p",
            str(IPERF_PORT), "-R", "-b", str(bandwidth), "-P", str(
            num_streams), "-w", str(window_size_bytes), "-Z", "-O",
            "3", "-C", "reno", "-t", str(duration), "-J", add_option,
            length_option_a, length_option_b],
        stdout=subprocess.PIPE,
        check=True,
        text=True,
    )
    result_down = json.loads(result.stdout)
except subprocess.CalledProcessError as cpe:
    print(cpe)
    result_down = None
```

Code 4: Starting the iperf3 download measurement in a subprocess on the operating system: -c denotes this process as the client, -R denotes that his is a download test (without it, it would be upload), -Z will make iperf3 use Zerocopy, which reduces CPU load, -O lets iperf3 omit the first 3 seconds of data (which are usually not used), -C tries to set the linux TCP congestion algorithm. add_option switches between UDP/TCP. length_option_a and length_option_b are for sending differently sized datagrams/segments.

Since iperf3 was called here with the -J option, a granular report in JSON format, split by seconds and streams, is generated after the completion of the measurement, which can be further processed by the program. This report explicitly refers to the data processed by the iperf3 process on the client. This means that for upload measurements that are performed with UDP, the data of the server must be used, since the client will most likely send more datagrams than the server will receive. On the client side, one would otherwise see all packages, whose transmission was tried, and not only those, which were received successfully. This is realized via the `--get-server-output` option, which allows the client to retrieve the iperf3 server statistics via the control connection after the measurement from the server is complete:

```
print("Start Measuring Upload from Client")
# Get the json from the server (via --get-server-output), as the json
from the client will not represent the correctly transferred data
try:
    result = subprocess.run(
            ['/usr/bin/iperf3', '-c', str(IPERF_SERVER), "-p",
            str(IPERF_PORT), "-b", str(bandwidth), "-P", str(
             num_streams), "-w", str(window_size_bytes), "-Z", "-O",
            "3", "-C", "reno", "-t", str(duration), "-J", "--get-
            server-output", add_option, length_option_a,
            length_option_b],
        stdout=subprocess.PIPE,
        check=True,
        text=True,
    )
    result_up = json.loads(result.stdout)
except subprocess.CalledProcessError as cpe:
    print(cpe)
    result_up = None
```

Code 5: Starting the iperf3 upload measurement in a subprocess on the operating system

After the two measurements have been performed a JSON object is now available, which contains the throughput recorded by iperf3 broken down by second, as well as some statistics about lost packets, whereby one evaluation per stream is possible, since the JSON contains an array with all used streams. For example, one second in a stream is encoded like this:


```
{
    "socket": 9,
    "start": 0,
    "end": 1.000072,
    "seconds": 1.0000720024108887,
    "bytes": 8992080,
    "bits_per_second": 71931460.76140644,
    "jitter_ms": 0.1281063357932468,
    "lost_packets": 50922,
    "packets": 57132,
    "lost_percent": 89.1304347826087,
    "omitted": true,
    "sender": false
}
```


Code 6: JSON object containing the data of the first second of stream nine of an iperf3 UDP measurement and information about the transmitted data. With UDP most of the transmitted data is lost (Target bandwidth was chosen much higher than actual throughput on the NIC).

As soon as the measurements are completed the two background processes for the latency measurement and the channel parameters are now also terminated. The second-by-second data of these background processes (see chapter 5.2.2) are now first stored locally as files together with the throughput data. This ensures that the measurement data is not lost even if it is not possible to upload the data to the Influx database. Then the data will be sent to Influx, if this is possible. If no connection can be established, e.g., because there is no connection to the Internet, no upload will take place and the program will end prematurely.

5.2.2 interface.py

The interface.py is the subroutine that monitors the channel, transmission and interface parameters in a background process. The following data is collected every second, provided that the interface provides this data (not all parameters can be retrieved with every configuration of the interface):

- MAC address of the associated AP
- Number of lost beacons of the AP
- Number of received beacons of the AP
- Bytes received by the AP and sent to the AP
- Number of frames dropped for various reasons
- Total number of transmit retries
- Negotiated bitrate in transmit and receive channel
- Channel ID
- Channel width
- Extended Service Set ID (ESSID)
- Frequency of the channel
- Link Quality (Intel-specific value (no further manufacturer specifications), between 70/70 (maximum) and 0/70 (minimum)
- Interface Mode (e.g., "managed" for Associated)
- Power management status
- Total bytes transmitted and received
- Guard Interval Type
- MCS index for transmit and receive channel
- MCS type (HT, VHT or HE, defined by the standard version)
- Number of Spatial Streams

- Receive signal strength of the AP signal
- Configured transmission power of the client

Originally, additional parameters such as transmission times and timing, and the negotiated modulation (OFDM/OFDMA) were to be retrieved, but this could not be realized via the Intel driver and this data could therefore not be retrieved with the interface used.

6 Test and Evaluation

The research questions can be divided into two parts: Evaluation of the software solutions for the management of IoT devices and the evaluations regarding Wi-Fi 6E and the technical capabilities of the available hardware. In this chapter, a functional comparison of the management software solutions is performed first. Since a client was only created for ThingsBoard (the prerequisites are not given for Thinger.io, as described previously), only ThingsBoard will then be discussed with regard to performance and operation with Wi-Fi 6E-capable devices.

In the following section, the performance of the Wi-Fi 6E hardware is examined in more detail and evaluated in terms of quality of service. Various measurement scenarios are described and analyzed for this purpose.

6.1 Comparison of the Management Software

ThingsBoard and Thinger.io differ in the characteristics and implementation of the basic functions of IoT management software. In the following, a direct comparison between ThingsBoard and Thinger.io is therefore presented for each of the basic functions.

6.1.1 Features and Functionalities

The features of IoT management software regarding device management can be summarized in subgroups like presented in chapter 2.3: Provisioning, authentication, configuration, control, monitoring, security, diagnostics and up-to-dateness. A comparison table was also created in advance, which assigns the individual subgroups to a point system in order to select the software to be considered in this work. This table can be found in Appendix A.

a) Provisioning

As part of provisioning, a device should be able to set itself up independently: It should be able to register itself independently with the platform and retrieve configuration data that it requires for further operation (e.g., access data). This can be done either via a predefined image for the device, or via a script that performs the necessary steps at system startup (automatic registration and preconfiguration). Only Thinger.io offers direct integration of the platform as a C

client library [34], for ThingsBoard this is only provided via defined HTTP API interfaces (which can then be called in a script) [35]. Automatic registration of devices is also possible on Thinger.io via an API, but this is provided by a plugin and is not usually part of a Thinger.io instance [29]. Without this plugin, provisioning must be triggered via the frontend by manually creating the device and so-called "device credentials". With ThingsBoard automatic provisioning is possible directly via a data pair: The device provision key and the device provision secret - including automatic assignment to a device group. A basic configuration can then also be directly retrieved by the device, including the latest firmware and software versions, should an update be directly available. Both software solutions show the (registration) status of the devices in the web interface [36] [37].

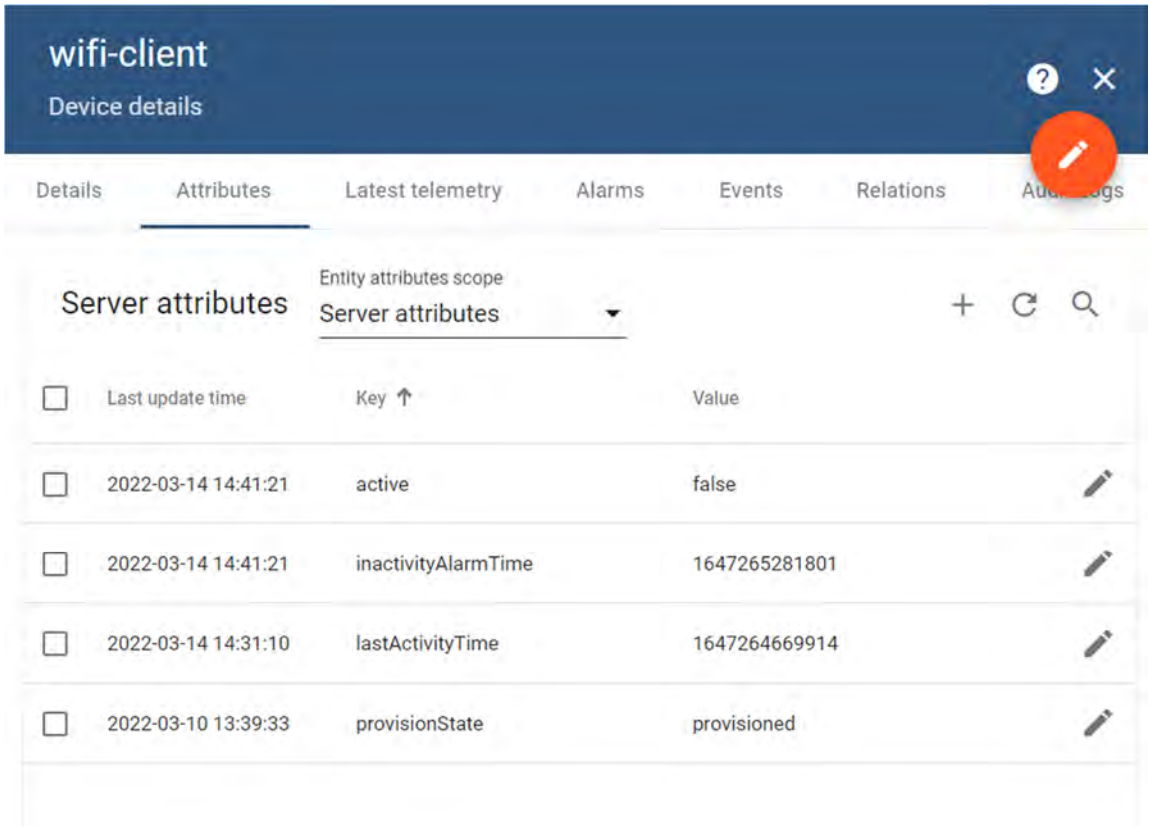

Figure 10: ThingsBoard showing the current state of a registered device "wifi-client". The device reports back when it successfully registered and the server will report connectivity information via the server-side attributes.

What is problematic in both cases is finding the management server: If the WLAN preconfigured on the devices is not available, the client cannot retrieve a configuration from the server (which cannot be reached), which may contain a different SSID-PSK combination for connecting to the server. However, this is only a limitation for WLANs that perform authentication via pre-shared key. This can

be circumvented, for example, by the clients basically using a certificate as part of EAP-TLS-based authentication. However, this is a restriction that does not result from the software.

Overall, registration to a ThingsBoard server is technically possible without user intervention once a device group has been created. With Thinger.io this can only be integrated via plugin, and this plugin is currently no longer available. So ThingsBoard is clearly preferable here.

b) Authentication

For authentication, it is important that the devices can uniquely identify themselves to the server. Thinger.io regulates this via the so-called Device ID, a string that globally uniquely identifies the device. This can be defined by the user himself [38]. In ThingsBoard there is also a device name, by which the device can be identified, but in addition each device is also assigned a device ID, which here, in contrast to Thinger.io, is a UUID (this is structured according to the Distributed Computing Environment (DCE) specification [39]). Other entities in ThingsBoard, such as Customer, are also identifiable via a UUID and can also be managed via the API. So here the implementation is consistent.

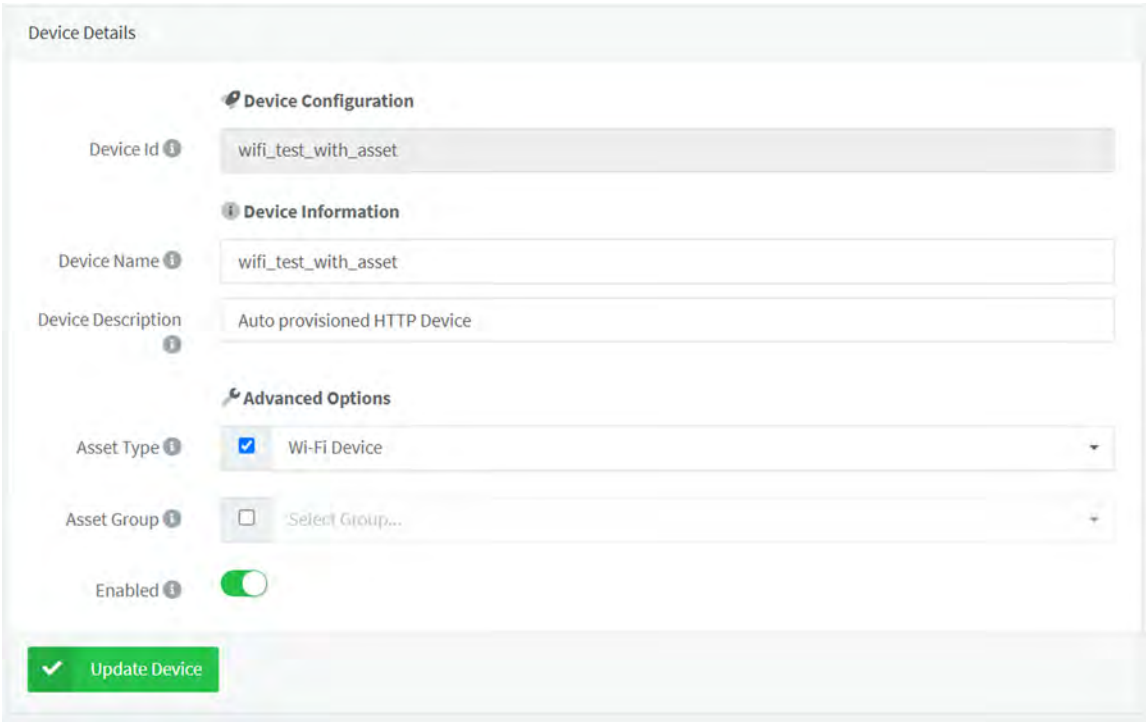

Figure 11: Thinger.io allows for a string without spaces as device ID. No two devices can use the same ID.

Both ThingsBoard and Thinger.io rely on JWT with Authorization Bearer Token for API authentication [40] [41]. This means that a user must first authenticate himself to the API with his access data before further API endpoints can be accessed via a bearer token. ThingsBoard allows sending telemetry data without JWT, but with a device token (or alternatively via X.509 certificate), which is static and does not need to be renewed [42]. However, since, for example, the device itself can only change device attributes with JWT, the device must still have classic access data (e-mail address and password) with which the device can authenticate itself to the API. This contradicts the basic idea behind the device tokens. Both softwares do not allow the device to authenticate itself directly via its device identity. The authentication parameters can be viewed in the frontend and can also be changed there (changing the token and the user credentials is possible).

c) Configuration (Over-the-air programming):

Over-the-air programming refers to methods of distributing software updates, configuration settings and sometimes key material to the target devices via a wireless interface (Like WLAN). Here it is also important that the devices can also communicate their current configuration and state to the management platform. This allows the identification of the devices and their configuration state.

ThingsBoard allows devices to update their attributes directly from the platform while also supporting updates by devices towards the platform [43]. Thinger.io also allows setting and retrieving the configuration (called "Device Properties" there), but this is only possible if user data (such as throughput measurement data in the context of this work) is not to be written to a bucket alternatively. Only one of the two options can be used exclusively via the API callback [44]. This is a strong restriction by the API and limits the usage options here. If we disregard this restriction there is another one: The API also only allows writing to and reading from a single "device property". Thus, in the given case, at most all configuration options of a device can be stored as one JSON object (or array), and no granular partitioning is possible, which is possible with ThingsBoard.

With ThingsBoard it is even possible to passively maintain configuration changes (persistent polling with timeout via HTTP GET). This is especially useful for changes that have to take effect immediately in case of doubt and cannot wait for

a polling interval: e.g. in case of necessary changes to the radio parameters of the Wi-Fi interface (in order not to lose the connection to the device) these parameters could be permanently requested by the device.

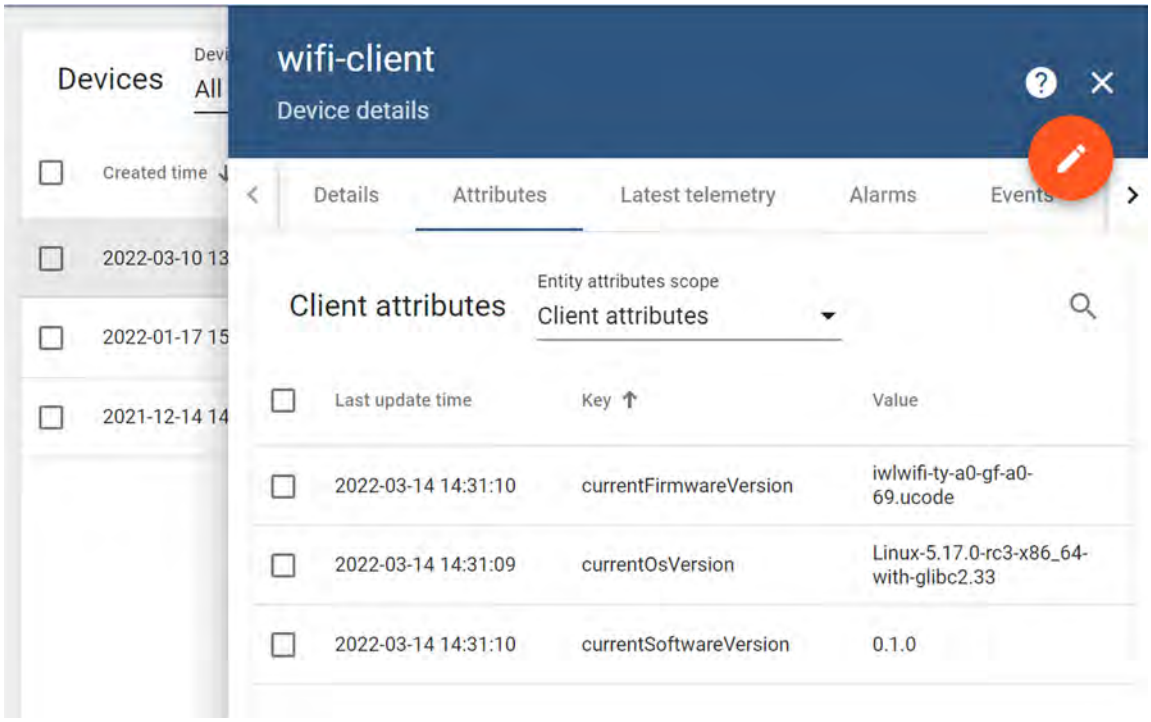

Figure 12: ThingsBoard shows device attributes either via a dashboard widget or the user can navigate to the device information page shown here.

Additionally possible with ThingsBoard, but limited to the "Device Properties" (i.e. the configuration) with Thinger.io, is the possibility to roll out software and firmware updates centrally for several devices in addition to the configuration. See section h) “Up-to-dateness” for more details. Device grouping is possible with both solutions and the current configuration of the devices can be viewed in each case. The clear advantage here lies with ThingsBoard, especially since no other data can be transferred from the devices to the management platform when using the configuration feature in Thinger.io.

d) Control

Thinger.io and ThingsBoard follow different approaches here: While Thinger.io lets the device itself determine which control options are available, ThingsBoard relies on remote procedure calls (RPC), which are then evaluated and executed by the device accordingly.

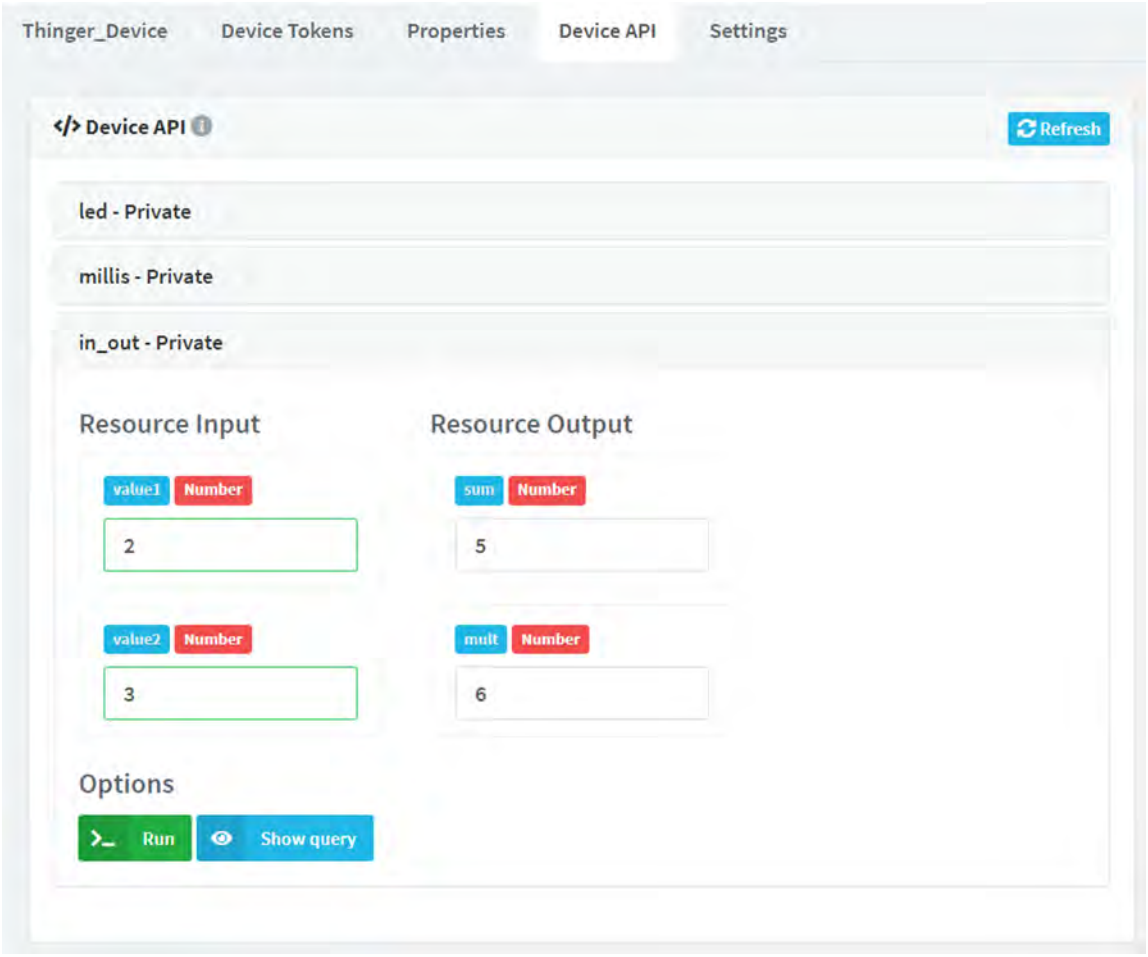

Figure 13: Devices can be configured to have input and output resources. On input resources the data can be manually sent to the device (Run button) and the outputs will be computed. Image taken from [45].

Thinger.io enables the device to define internal controllable input and output options via the so-called Device API, which are then discoverable in the web interface and can be addressed there [45]: Commands can be sent to the device and results can be returned. It can also be used to map more complex functionality, such as changing the device status (e.g., restarting or disconnecting the network connection) or starting measurement series.

ThingsBoards RPCs are also very flexible: It is possible for the client to make requests to the platform via RPC and for the management platform to respond with data (e.g., to retrieve information about whether the measuring station is currently occupied by another device) as well as sending calls by the platform to the device. For information on the use of this feature, please refer to the chapter 5.1.1.

Overall, both approaches can be used to map sufficiently granular control to trigger commands such as device state changes, updates or moving an actuator.

e) Monitoring

Devices, especially devices equipped with sensors, generate data for which monitoring is useful. Thinger.io offers the two known possibilities to store data in the system, either as "Device Property" or in the Data Buckets, which record time series data similar to a database table [44] [25]. Devices can regularly send data to the platform for this purpose [46]. Both options can be used to record system metrics such as CPU utilization or local hard disk usage, for instance. For regular measurement data, however, as in the case of this work, only a data bucket for large amounts of data comes into question. This data can also be presented to the user in dashboards. It is not possible to notify or alert the user in the event of unusual values or limit values being exceeded.

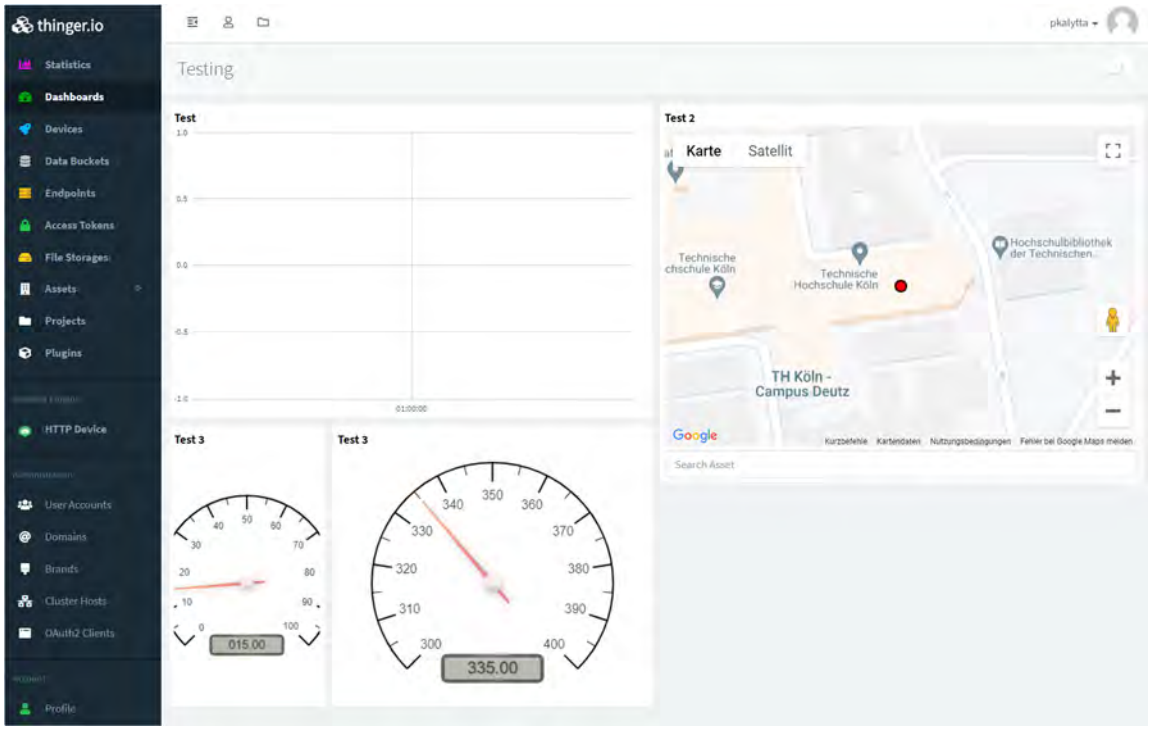

Figure 14: Thinger.io dashboard can show device properties and data from data buckets via different widgets.

ThingsBoard, on the other hand, enables such alerting by using the rule engine as mentioned in the previous chapter [47]. There, for incoming data, a client can individually define at which limit values or according to which logic alarms are to be triggered.

Clients can send their system metrics and other telemetry and measurement data to the telemetry upload API. For this the client only needs the appropriate access token [43]. Overall, the ThingsBoard approach is more robust because custom rules can be defined for monitoring in the rule engine and a central collection point for incoming data is available with the telemetry API.

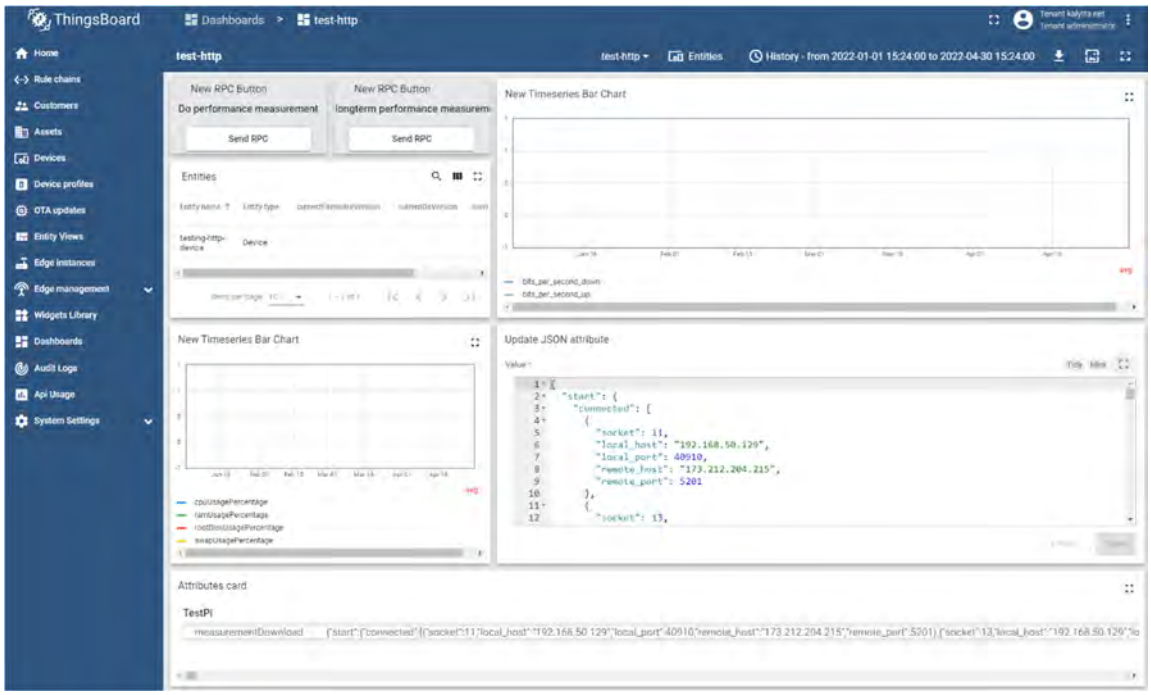

Figure 15: ThingsBoard dashboards can get data from device attributes, the internal rule chain or from the telemetry data. Also, Remote-Procedure-Calls can be directly triggered from a dashboard.

f) Security

To be able to guarantee the security of communication the transmission channel for information must be secured and there must be authentication for the devices. Regarding the operation of the user interface the rights and options of the users should be limited by role-based or attribute-based access control.

Thinger.io does not specify in the documentation whether the http connections are encrypted. In the test environment, however, it could be determined that the environment is generally run with HTTPs support enabled by default. This allows encrypted transmission of configuration and measurement data, which is particularly necessary for wireless communication. The devices must also authenticate themselves to the platform when accessing the API. Authentication via JWT Authorization Bearer Token is offered for this purpose:

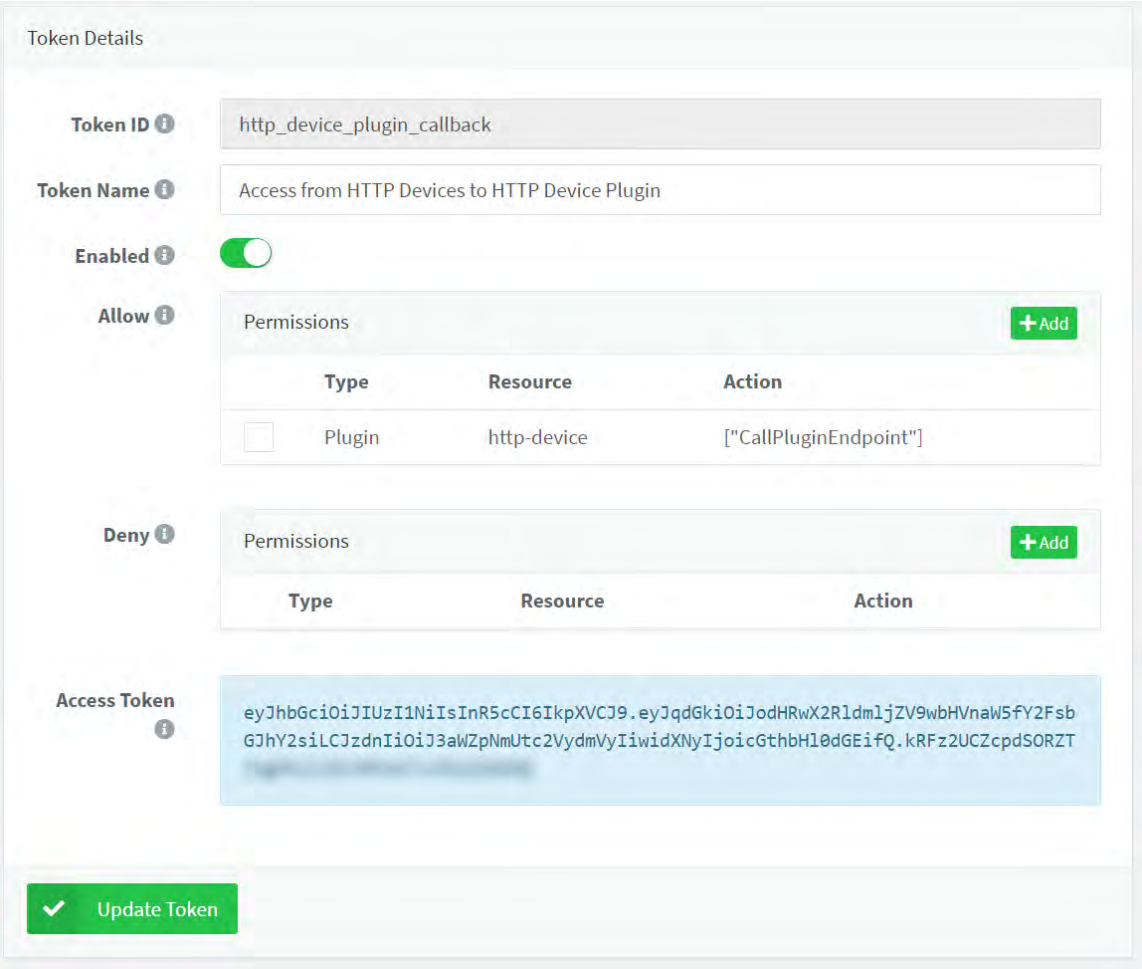

Figure 16: The Thinger.io web interface allows for access control for devices via tokens: i.e. a token can be specifically created to only allow write access to one data bucket.

What is not possible in the tested version of the software is securing through role-based access control (RBAC) - this is only possible in the Professional or Enterprise Edition of Thinger.io [48]. In the tested version only one user account was possible. Additionally, no multi-tenant capability is given. According to the documentation, this is possible by separating into projects in the Professional or Enterprise Edition of Thinger.io [49].

ThingsBoard takes a similar approach to Thinger.io regarding securing connections to connected devices: After creating a ThingsBoard server as described in the documentation, encrypted communication is not yet possible at first, but HTTPs can be enabled via the ThingsBoard configuration file, both for the web interface and the API [50]. ThingsBoard also relies on token-based authentication for endpoints (but not with JWT) but its own token concept that integrates directly with the API URL, so that a token co-defines the API endpoint [42] [43]. Likewise, a granular role-based access concept for users is only possible

in the Professional Edition [51]. In the tested free Community Edition, a distinction can only be made between System Administrators (can create and delete tenants), Tenant Administrators (can manage devices, dashboards, customers and other entities) and Customers (can read dashboards and control devices). In contrast to Thinger.io, however, any number of users can be assigned to these roles. It also results from this that already in the free version of ThingsBoard a multi-tenant capability of the software is given. Here ThingsBoard is clearly better suited for larger deployments.

g) Diagnostics

Thinger.io records the general connectivity of a connected device, i.e. when communication was last established and whether the device is currently sending data. It is also possible to access the server logs in which all accesses are recorded centrally. However, this is not possible from the web interface, but only directly on the server for an administrator. No audit logs are created for the user interface.

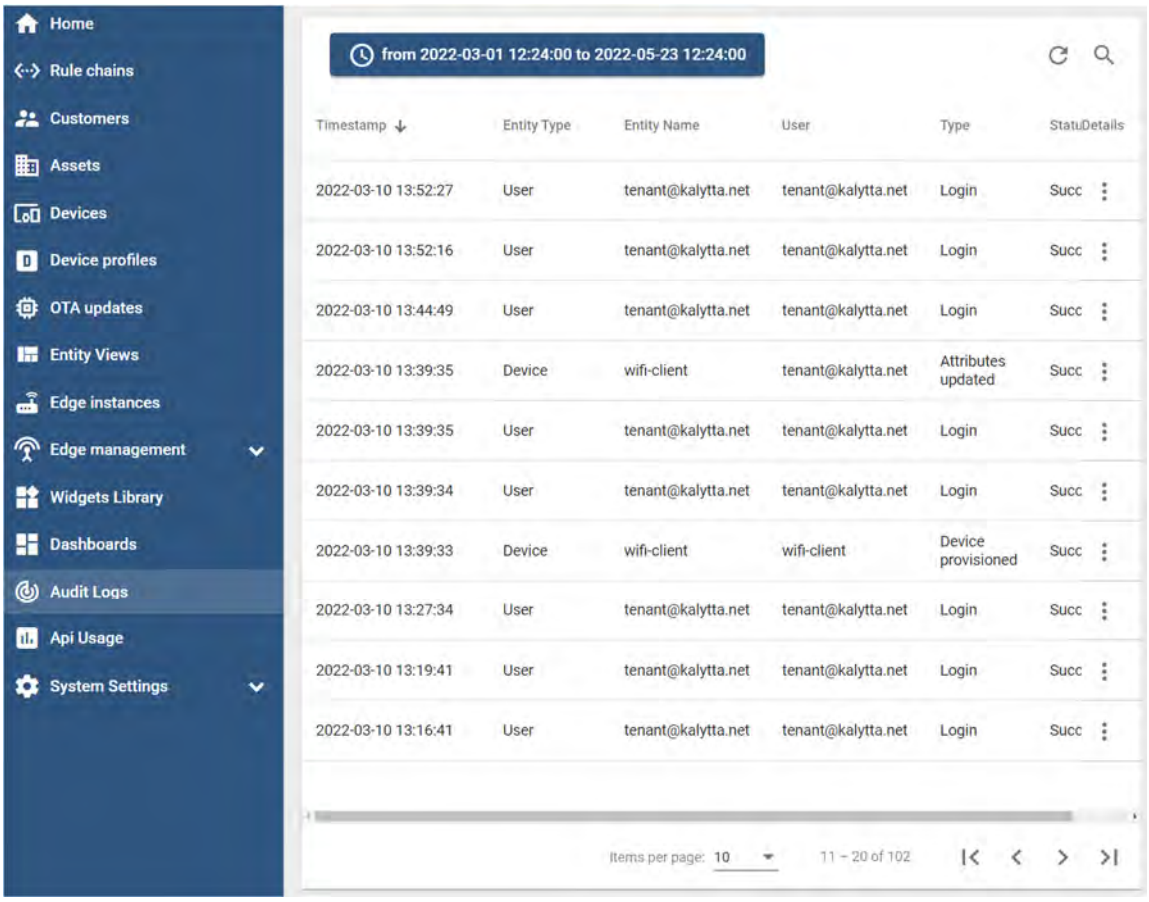

Figure 17: ThingsBoard shows user and device generated events in a "Audit Logs" tab on the webinterface.

ThingsBoard also enables device status recording, but in two different ways: On the one hand, it records when telemetry data was last sent - this can be viewed per device for diagnostic purposes. On the other hand, the API also offers the option of setting attributes for the device's status (e.g., CPU utilization or firmware version), which can then be evaluated centrally [52]. ThingsBoard does not provide central logging for all data in the basic configuration, but rule chains can be used to perform granular logging through the "log" module for data in a rule execution. This data is then written to a central server log that, like Thinger.io, is not accessible in the web interface. Additionally, ThingsBoard provides audit logs in the web interface. These logs record events such as user logins, device registrations and changes to attributes, dashboards and entities.

h) Up-to-dateness

ThingsBoard, in addition to updating attributes on the server side (which the client can retrieve) and the ability to deliver information and updates to devices via RPC, also provides a third way to ensure actuality, especially of installed software: Over-the-Air (OTA) update packages:

Both software and firmware updates can be stored in the administration interface and assigned to a device profile. Devices in this profile can then compare the assigned version with the installed version and download the newer version if there is a mismatch. The administrator can either specify an external download link (good for e.g. content delivery networks), or the update can be downloaded directly from the ThingsBoard server. The process is also then protected from integrity problems by a checksum. System updates can therefore be managed well centrally, rollbacks are also possible by simply assigning another version of the firmware or software for a device profile.

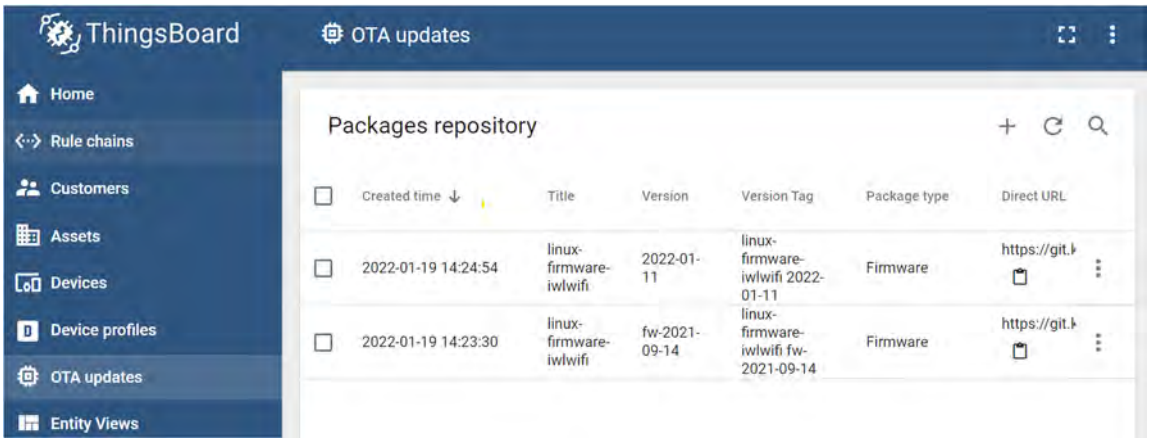

Figure 18: ThingsBoard enables the tenants to centrally manage software and firmware updates for single devices or for bulk updates to a device group. This is useful to update a lot of devices at once.

Configuration backups are not possible via this functionality, however. In this case, ThingsBoard must fall back on RPC or Device Attributes, which can be used to store several versions of a configuration on the server side for example. However, the import of an older configuration version is not automated in this case.

Thinger.io only supports firmware update for various microcontrollers, not for generic devices connected via HTTP API [53]. The functionality is implemented in the case of microcontrollers via a plugin in Visual Studio Code, a management of multiple versions must be performed as here in Visual Studio Code and not in the web platform of Thinger.io by the user. Performing configuration backups for devices and rollbacks via Thinger.io is not currently possible.

Overall, the question of which of the two softwares is better used to manage Wi-Fi 6E-enabled devices can be answered clearly: Thinger.io does not provide some of the necessary features provided by ThingsBoard:

The automatic provisioning of the devices including the provision of configuration data as well as the latest firmware and software versions is much easier to realize via ThingsBoard. Especially for software updates, Thinger.io would have to rely on external sources, since provisioning by Thinger.io is not only possible for microcontrollers. In ThingsBoard, the configuration of the devices can be mapped by attributes, from which, for example, measurement data can be separated by supplying them to the platform as telemetry data. This is another clear advantage

over Thinger.io, where this separation also exists, but only one of the two data types can be sent to or retrieved from the server due to the API limitation. ThingsBoard is therefore preferable because of the available features and functions, since a connection of generic Linux Wi-Fi devices can be implemented here more completely and clearly.

6.1.2 Structural Differences

Particularly noticeable is Thinger.io's clear reference to microcontrollers: The well-known microcontrollers from Espressif ESP32 and ESP8266 as well as the Wi-Fi, Ethernet and GSM-supporting controllers from Arduino are explicitly mentioned as supported [54]. There are also coding examples for these platforms in the documentation. This also differentiates Thinger.io from ThingsBoard in that ThingsBoard does not directly specify devices that are supported here but defines a number of protocols that any device can use to communicate with the platform: In the free community edition these are MQTT, CoAP, HTTP, SNMP and LWM2M [55]. Each device group can be connected via one of the protocols. An existing MQTT infrastructure can even be connected via external MQTT gateways..

The approach to multi-tenant capability is also clearly different. ThingsBoard is shipped with this feature and at least one tenant must be created, even if no other tenants are served on the system. There are also versions of Thinger.io (like the one used for this work) that do not have this feature enabled at all.

Thinger.io uses JWT authentication for every data transfer between the platform API and the devices, while ThingsBoard differentiates between the tokens used for telemetry or provisioning, for example, and the username/password access data required for writing shared attributes, for instance. In the case of ThingsBoard, the uniform control of the devices via RPC should be emphasized, while in the case of Thinger.io, the API is not complete in this respect and offers the features and shortcomings described in chapter 4.1.3.

6.1.3 Performance

Sending data by connected devices to ThingsBoard never caused any problems in several tests. Even larger amounts of data generated by the throughput measurements over several minutes and then sent collectively to the platform

could be processed without problems. No statement can be made here regarding Thinger.io, since this functionality could not be tested due to the lack of a basis.

ThingsBoard states in the documentation that firmware and software updates can be delivered to 100 connected devices simultaneously by default [56]. This can lead to problems in reality: Several hundred TCP sessions heavily loaded by downloads may be impossible to handle with weak hardware. However, the number of simultaneous downloads can be adapted to the hardware conditions in the configuration of ThingsBoard. Problematic in the current setup was the limitation of software and firmware packages stored on the ThingsBoard instance to a maximum of 2 gigabytes. Updating the complete firmware stack on the Linux clients was therefore not possible from the local instance, but only via direct download from the Linux kernel repositories.

ThingsBoard's web interface scores 56 out of a possible 100 points in Google's Lighthouse Performance Test. Points are deducted for heavy JavaScript usage and the lack of text compression. Furthermore, the browser is not prompted to cache resources. Thinger.io scores much better with 78 out of 100 points in Lighthouse, where only the lack of HTTP/2.0 support and the missing cache policy (like ThingsBoard) are criticized. Both platforms react sufficiently fast to user inputs in the frontend or show a loading animation when loading for a longer time.

6.1.4 Operation/Frontend

Thinger.io offers after the login screen directly an overview page with the statistics of the currently connected devices and the transferred data:

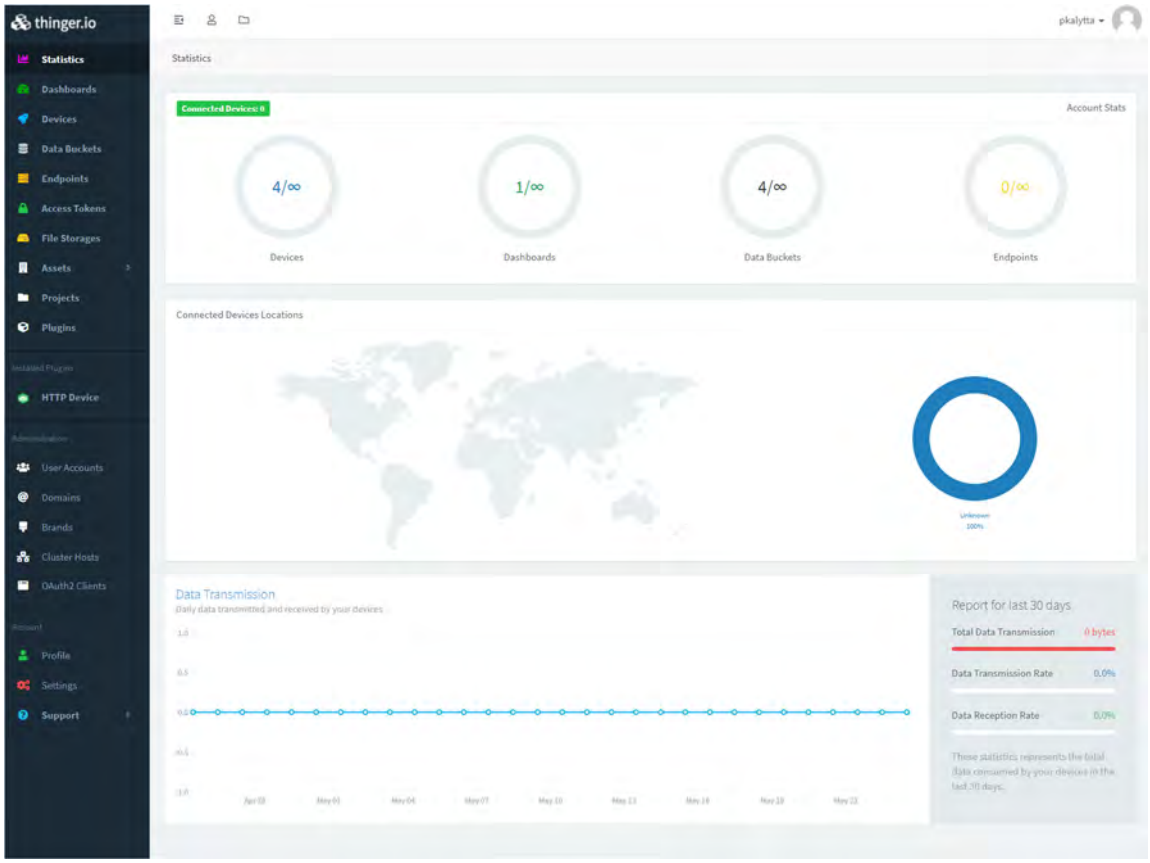

Figure 19: The start page for Thinger.io shows the number of connected devices, dashboards, data buckets and other endpoints over a world map, that shows currently connected devices that send their coordinates. Below that, the data transmissions for the last thirty days are shown.

On the left side Thinger.io's web interface offers a list of configuration options grouped in menus: Dashboards, Devices, Data Buckes, Endpoints, Access Tokens and File Storages are listed at the top and can be listed, edited and created from there. Further down in the menu is the plugin management, the configuration options for the installed plugins (here: The HTTP-Device Plugin) and the administration options. The admin options are not usable in large parts although they are displayed. This is due to the fact that the license used does not allow the creation of user accounts, but the option is still displayed.

Individual sub-items in the menu, e.g. the Devices sub-item, then always offer further configuration options for this menu item in the right part of the window:

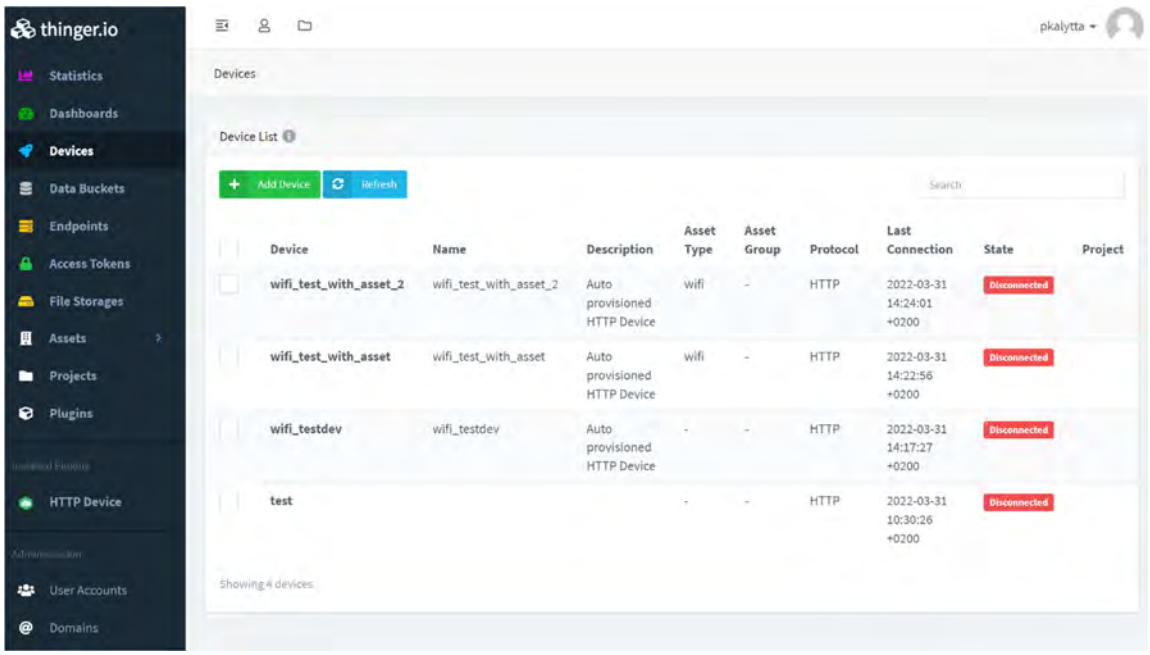

Figure 20: Under the menu point "Devices" Thinger.io will show a list of all configured devices and allows for creation of new device configuration.

ThingsBoard offers two separate web interfaces depending on whether one is working as a system administrator or as a tenant. The system administrator can change system settings and manage tenants and tenant profiles from his interface:

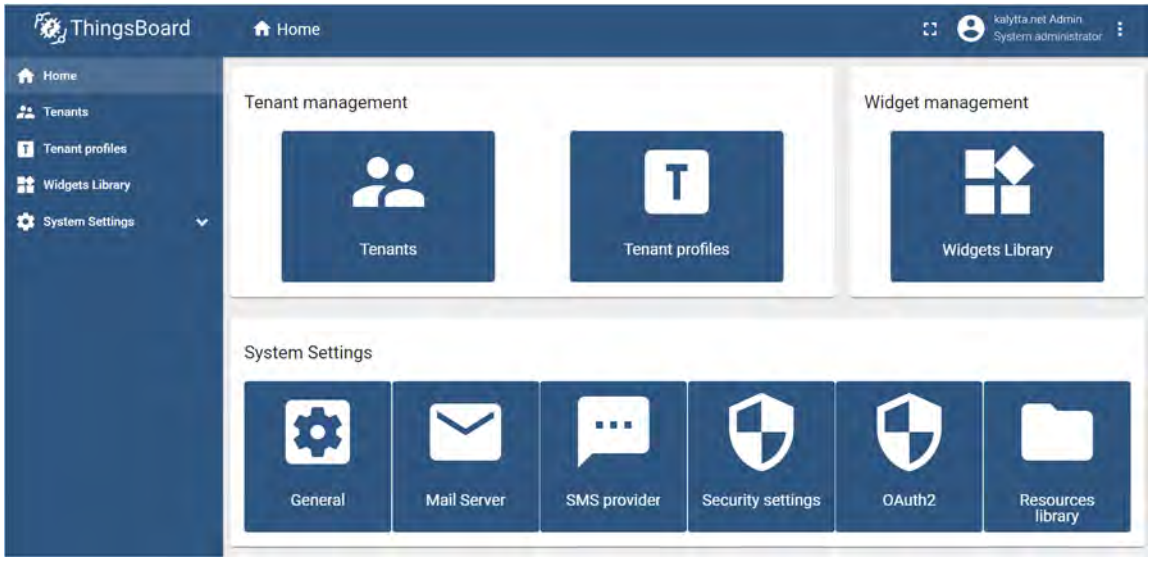

Figure 21: Web GUI for the ThingsBoard system administrator after login. It is similar in design to a tenant web GUI but shows different options in the menu on the left.

The menu structure is also integrated into the left part of the browser window in a column. The sub-items are not divided into groups (except for Edge management and System settings), which makes it difficult to keep track. Only on the start page (Home) a grouping of the individual sub-items as tiles in a grid can be found:

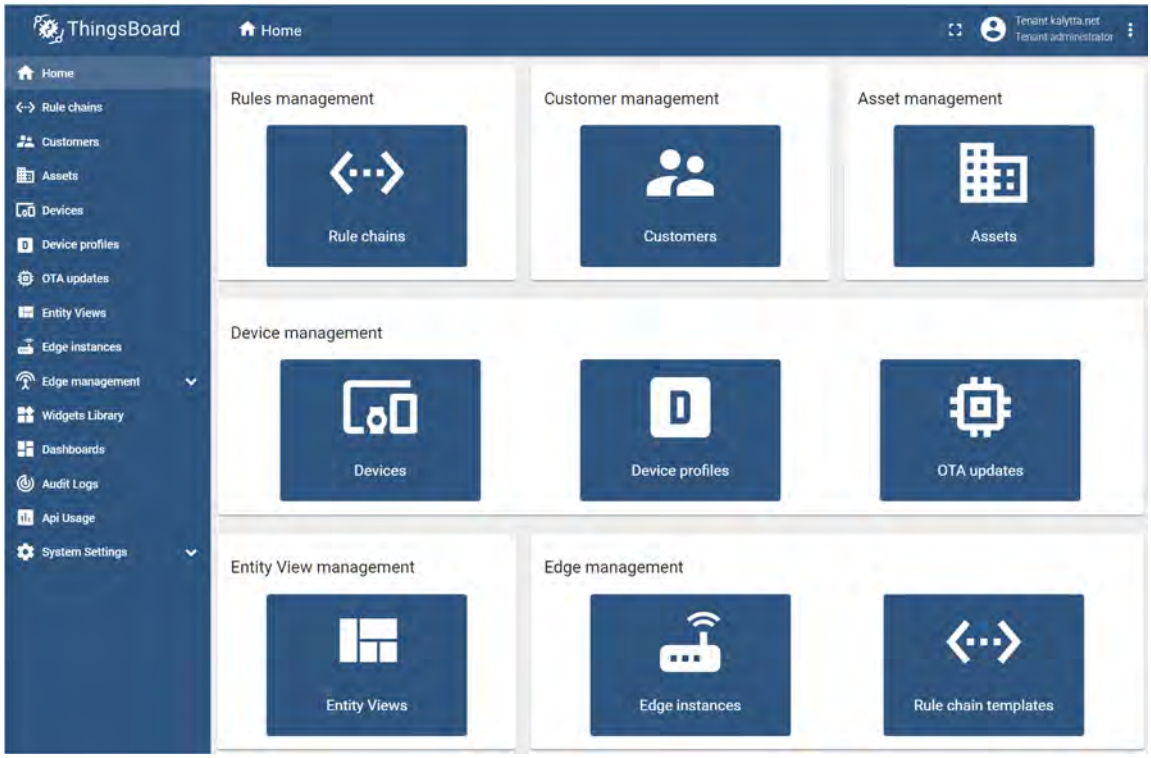

Figure 22: The ThingsBoard dashboard start page shows large tiles that allow navigation to the specific configuration options, dashboards and other parts.

Similar to Thinger.io, further configuration for a topic can then also be realized via the individual sub-items. Devices can be created, configured and deleted via the "Devices" subitem:

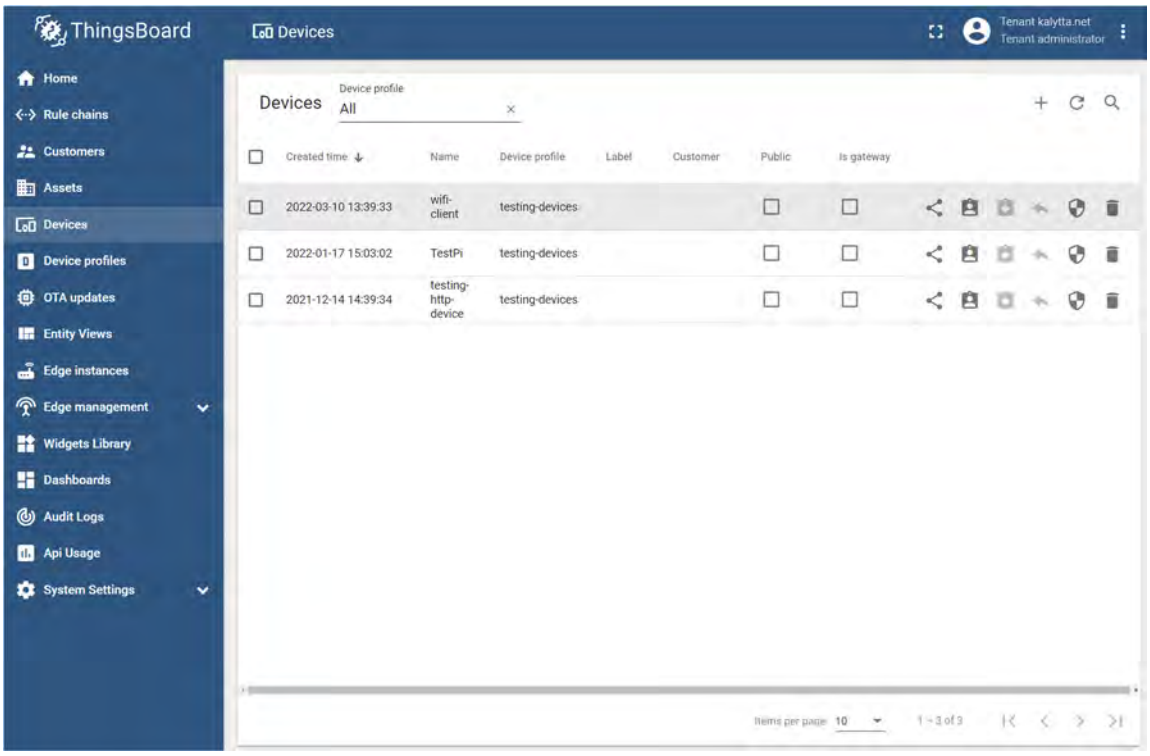

Figure 23: The menu point "Devices" will show all configured devices on the right part of the browser window. Clicking on a device will open a slide-in window with the devices configuration options.

Thanks to the rule chains, ThingsBoard can perform any number of complicated dependencies and data transformations via the front end: Since JavaScript can be used as a scripting language, sent JSON data, for example, can be fully processed. There are also already predefined so-called "nodes" with which data can be enriched, filtered and transformed and actions can be triggered.

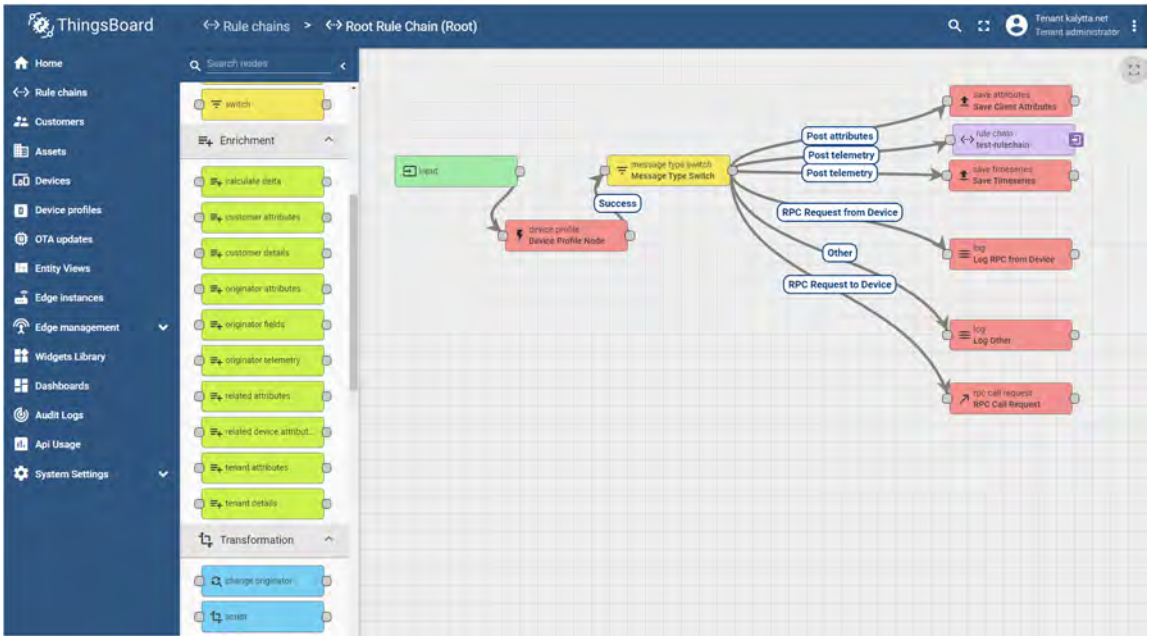

Figure 24: ThingsBoard Rule Chains can filter incoming data, transform outgoing data and react to it, i.e. by logging it, sending an RPC or generate an alert.

The web interfaces of Thinger.io and ThingsBoard are very similar in structure and operation: Both offer a menu structure on the left side and display the corresponding content on the right side (e.g., configuration options, lists of devices, tokens or dashboards or, if a dashboard is selected, the content of this dashboard. By specifying meaningful names in the menu (e.g., item "Dashboards" shows the list of configured dashboards"). Operation is intuitive even for inexperienced users. Thinger.io additionally relies on color delimitations (individual menu items have different colors), while ThingsBoard uses a corporate design here that has uniform design elements (blue background, white icons and font). In the ThingsBoard Professional edition a customization of this design is possible [57].

6.1.5 Managing Wi-Fi enabled Network Devices centrally

When it comes to managing generic Wi-Fi enabled network devices, such as Linux-based access points, metering stations or routers, ThingsBoard is clearly preferable to Thinger.io. ThingsBoard's HTTP API can be used in a hardware-agnostic way and provides a good basis for connecting arbitrarily complex devices to the management platform through auto-registration, the ability to send telemetry data to device-specific API endpoints and control via RPC. The data

can be further processed via the rule chains, e.g. can also be sent to the Amazon Cloud (AWS) or an external MQTT broker.

Thinger.io offers some functions only for the C client software specifically intended for microcontrollers. Functions that are actually necessary, such as saving measurement data and configuration data for a device at the same time, cannot be performed via the HTTP API.

The management of Wi-Fi 6E-capable Linux devices is thus only sensibly possible with ThingsBoard. As a management platform ThingsBoard offers sufficient configuration options to be able to cover a wide range of use cases and is therefore suitable for the use cases presented in this work (measurement of Wi-Fi 6E connections).

6.2 Evaluation regarding Wi-Fi 6E

Since it can still be assumed that the channels in the 6 GHz band are not being used by other subscribers at the time the measurements are carried out, there is the advantage that throughput measurements are hardly or not at all influenced by other radio transmissions. This means that the actual performance of the hardware used (AX210 & Aruba AP-635) can be viewed with greater certainty.

On the one hand, the selected hardware parameters are particularly decisive here, such as the channel bandwidth, where 160 MHz can be used in the 6 GHz band for the first time. This will be less relevant for IoT applications, since the maximum 2402 Mbit/s possible with OFDM modulation and two spatial streams will exceed the requirements of most IoT use cases. It is nevertheless interesting to look at what data throughput is possible at the transport layer (TCP/UDP) over a Wi-Fi 6E connection also in comparison to previously used Wi-Fi 6 (802.11ax) and Wi-Fi 5 (802.11ac) connections.

On the other hand, special attention was to be paid to the newly usable OFDMA modulation method (which is also used in LTE, for example). However, it was determined that OFDMA cannot currently be used with the available hardware (see chapter 6.2.8).

6.2.1 Hardware

The driver of the AX210 does not yet offer all configuration options that are usually available and can be manipulated by tools like iw. In particular, the modulation could not be forced explicitly (e.g. OFDMA could not be forced) and no fixed transmission bit rate could be specified. An error message was always generated according to the following specification:

```
SET failed on device wlan0 ; Operation not permitted.
```

This is a feedback from the Netlink interface that the driver or firmware has refused to change the parameter.

In addition, an error was found in the firmware's data feedback to the operating system: For the NIC, wrong limits regarding the transmit power within the selected regulatory domain are forwarded (22 dBm for all channels in the 2.4 GHz, 5 GHz and 6 GHz bands) [14]. However, the transmit power of the card could be reduced manually so that the limits could be met.

a) Influence of the channel bandwidth

In the 6 GHz band all possible channel bandwidths (20, 40, 80 and 160 MHz) could be successfully configured and data could be transmitted. All three available 160 MHz channels could be used in the tests, SSIDs on these channels were found during scanning of the AX210 and association was possible without any problems. However, the additional channels added with the 6 GHz frequency band increase the scanning interval to about 7 seconds in order to be able to scan all available channels in the three frequency bands.

b) Influence of the modulation parameters

The MCS indices 10 and 11 with 1024-QAM, which are newly possible with the 802.11ax standard, result in a further increase in the theoretical throughput on the radio interface. In the following measurement series it becomes clear that these MCS are also frequently selected under good conditions (high reception quality), but less so with 6 GHz. Likewise, the use of two spatial streams is also frequently added in TX, so that the gross data throughput negotiated by the card often corresponds to the maximum possible data throughput in transmit.

Another limitation of the card was found here: Depending on the NIC used (besides the AX210, the AX201 is also affected) only one spatial stream is possible in the RX. The NIC does not use the possible two spatial streams despite good channel parameters. In the lab measurements during this work, the behavior was primarily found when using the card in the 6 GHz frequency band (as a result, the transmit data throughput (upload) is often about twice as large as the receive data throughput (download)). However, other users report in the corresponding kernel bug report that this problem can also occur when using 802.11ax in the 5 GHz band [58]. This could not be observed in the laboratory measurements within the scope of this work. A bugfix proposed in the bug report does not bring any change.

However, these statements only refer to modulation with OFDM. OFDMA could not be tested.

6.2.2 Quality of Service and Performance

For 6 GHz and 5 GHz, the results are basically similar as long as the transmission parameters are the same. For example, if the channel bandwidth at 6 GHz is restricted to the 80 MHz possible at 5 GHz and care is taken to observe the limitation of the spatial streams. The diagrams shown in the following can be found enlarged again in the appendix.

a) Throughput in Wi-Fi 6E and Wi-Fi 6 (802.11ax)

In throughput measurement, one recurring phenomenon is particularly worth mentioning: In TCP throughput measurements with iperf3 single zero data points can be observed in some cases, which drop out of the usually expected measurement series. This behavior does not follow a recognizable pattern and is probably due to a limitation in the granularity of the measurement (acquisition of data points happens every second): When merging the data series from iperf3, these null values then occur. In a packet capture that was performed during such a measurement these zero transmissions are not included, but a continuous data flow is present.

Figure 25: Measurement over 6 GHz with 80 MHz channel width, TCP: Some data points in the upload are zero.

In general, however, fluctuations of varying intensity can be observed in the data series. Fluctuations in net data throughput are to be expected with radio transmissions, especially in the 5 GHz band, which may also be occupied by other participants.

b) Latency

For latency detection, the round-trip time (RTT) to the iperf3 server was also measured continuously during the throughput measurement. An interesting picture emerges, especially for the download to the client via TCP:

While stable RTTs are measured in the upload in comparison (the slight existing jitter is to be expected for a WLAN connection) an edge-like regular increase of the RTT (like a sawtooth curve) results in the download. This is an indication that the interface is much more heavily loaded in the upload, so that packets may collect in a buffer here before they are sent. However, this sawtooth-like curve is only observed in the RTTs of the latency measurement and does not seem to have a significant effect on the data throughput measurement:

Figure 26: Measurement over 6 GHz with 160 MHz channel width, TCP. Download with one spatial stream, upload with two spatial streams. Throughput reaches over 1 Gbit/s in this case.

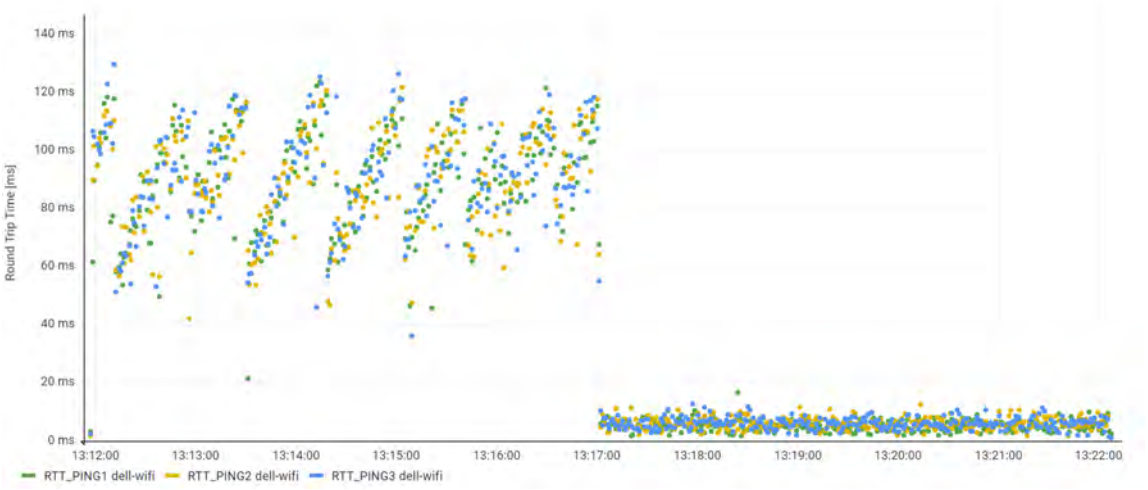

Figure 27: Corresponding RTT measurement. The sawtooth-like behavior of while downloading with high jitter can be clearly differentiated from the low jitter behavior while uploading from the client.

These specific fluctuations do not show up in the measurement with UDP. With UDP considerably longer RTTs occur (when comparing with the more stable transmit-part of TCP) but these are arranged in an edge-like manner:

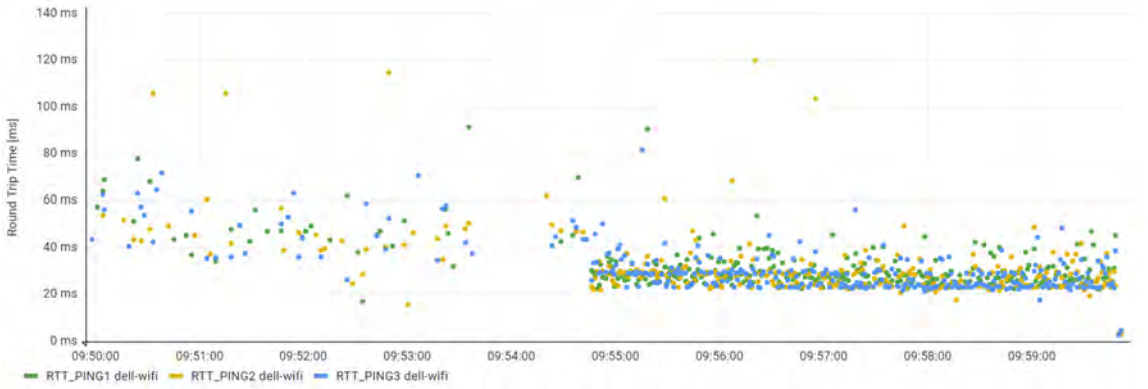

Figure 28: RTTs measured via ICMP Ping for a UDP measurement, the first half for download, the second half for upload from client.

c) Stability/Availablity (Longterm Measurement)

Individual measurements were performed for more than 12 hours in both the 5 GHz band (Wi-Fi 6) and the 6 GHz band to identify possible long-term problems with high continuous data throughputs. In both cases no problems were identified. Both the client-side transmit retries are in the negligible range (below 0.02% TX retries on all attempted frame transmissions) and the dropped frames in the receive, which are zero or close to zero in almost all measurements. Regularly, beacon frames, which were kept in a separate drop category, were dropped. This is probably due to remote access points whose beacons could not be completely

decoded correctly during scanning. The connection between the clients around the APs can be considered stable as long as the reception quality is good. In the distance measurements the reception quality naturally decreases as the distance increases, thus also the stability, which first manifests itself there in the reduction of the MCS and spatial streams and then in errors and dropouts in the data transmission.

6.2.3 Measurement scenario: Client-to-Client in one WLAN cell

If a measurement is performed within a cell (Single BSS) between two clients (or STAs), the data must still be transmitted via the access point with which both clients are associated. This means that in the next scenario (Chapter 6.2.4) we expect about twice the throughput compared to the values measured here.

Reduced throughputs can be expected here due to the channel being occupied by one of the clients in each case and when the frame is forwarded by AP to the other client, the maximum of which is primarily determined by two parameters: The respective negotiated MCS of the two clients. Even if one of the clients negotiates the maximum (e.g., at 80 MHz and two spatial streams with the short guard interval (0.8 µs) a maximum of 1201 Mbit/s gross data throughput) the MCS of the other client, which may have been chosen lower, can limit the throughput here. In such a scenario special care must be taken to ensure that both clients have the best possible connection to the AP, or at least a connection of approximately the same quality. Otherwise, one of the clients will inevitably restrict the throughput.

Figure 29: Measurement over 6 GHz with 80 MHz channel width, TCP. With two spatial streams in the TX, theoretical throughput is 1.2 Gbit/s, due to limits by the RX spatial stream (only one), real throughput is much lower (only about 230 Mbit/s)

In this measurement scenario this problem can be strongly seen in the fact that, as already described above, only one receive spatial stream is used for the clients at 6 GHz due to software limitations. Therefore, it is then irrelevant that the clients in the transmit can offer two spatial streams: The data throughput is reduced to about half of the theoretically possible maximum (defined by the MCS) in the receive, and even only about a quarter of the maximum in the transmit.

Since the use of two spatial streams in both directions is possible at 5 GHz correct communication via two spatial streams in both directions can be observed here. The comparison shows that the data throughput is approximately doubled, but there are also stronger fluctuations in the data throughput, presumably due to the higher channel occupancy:

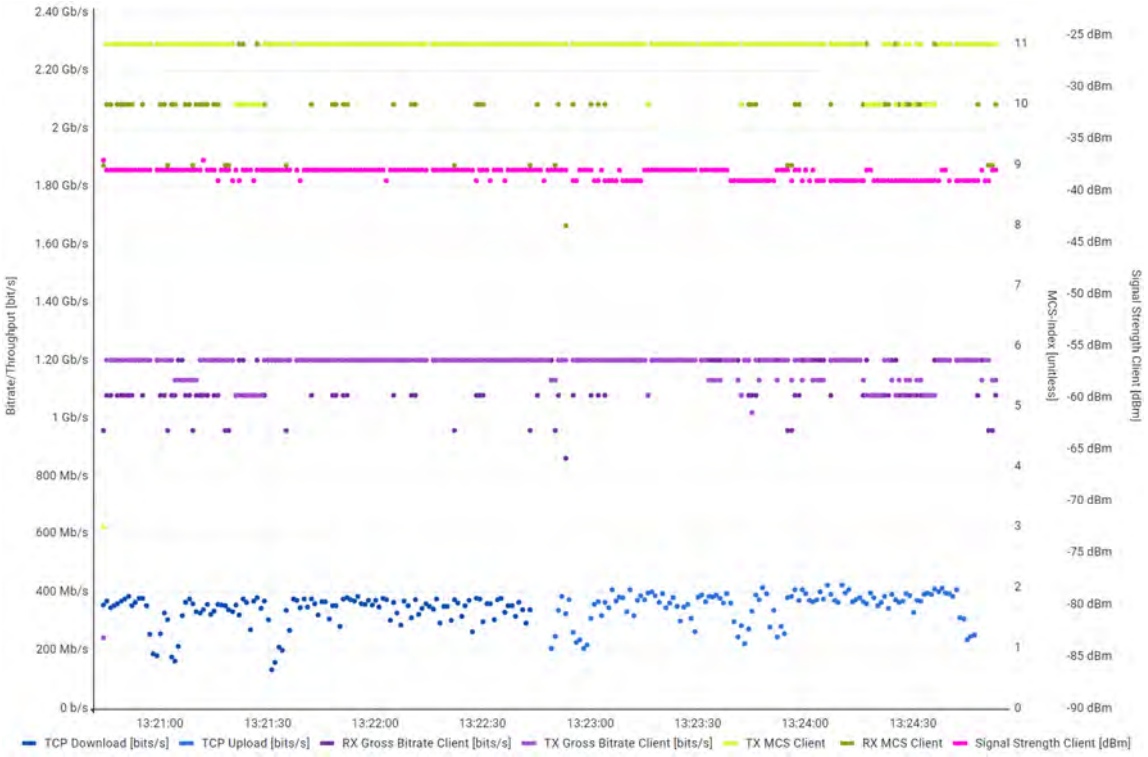

Figure 30: Measurement over 5 GHz with 80 MHz channel width, TCP. Throughput is nearly doubled with about 400 Mbit/s. The MCS flapped between 10 and 11, corresponding to 1080 Mbit/s and 1201 Mbit/s for both RX and TX.

6.2.4 Measurement scenario: Client-to-external-Server outside of the cell

When using the external server (connected to the AP via a cable connection) relatively interference-free communication between the AP and a client can now be considered guaranteed in the 6 GHz band. The practical maximum throughput that can be achieved can therefore be tested here at 160 MHz channel bandwidth. The following should be noted here: The maximum gross data rate is only achieved if:

a. The shortest guard interval is used (0.8 µs),
b. One of the widest channels is used (160 MHz),
c. The maximum MCS is negotiated between AP and STA,
d. Both possible spatial streams are used.

Not all the conditions mentioned here are always given for the following measurement series. Only one spatial stream was available in the download (RX) of the measurement client. If only one of the two spatial streams is used the throughput maximum for one spatial stream is tested. Additionally, the maximum MCS could rarely be negotiated with the AX210. Most of the time only an MCS index of 8 or 9 is possible. This also seems to be a limitation due to the Linux system or the driver used, since the maximum MCS is possible under Windows under the same conditions (only the measurement software could not be used there).

TCP has generally a lower data throughput than UDP due to mechanisms like congestion control, out-of-order delivery/retry etc.

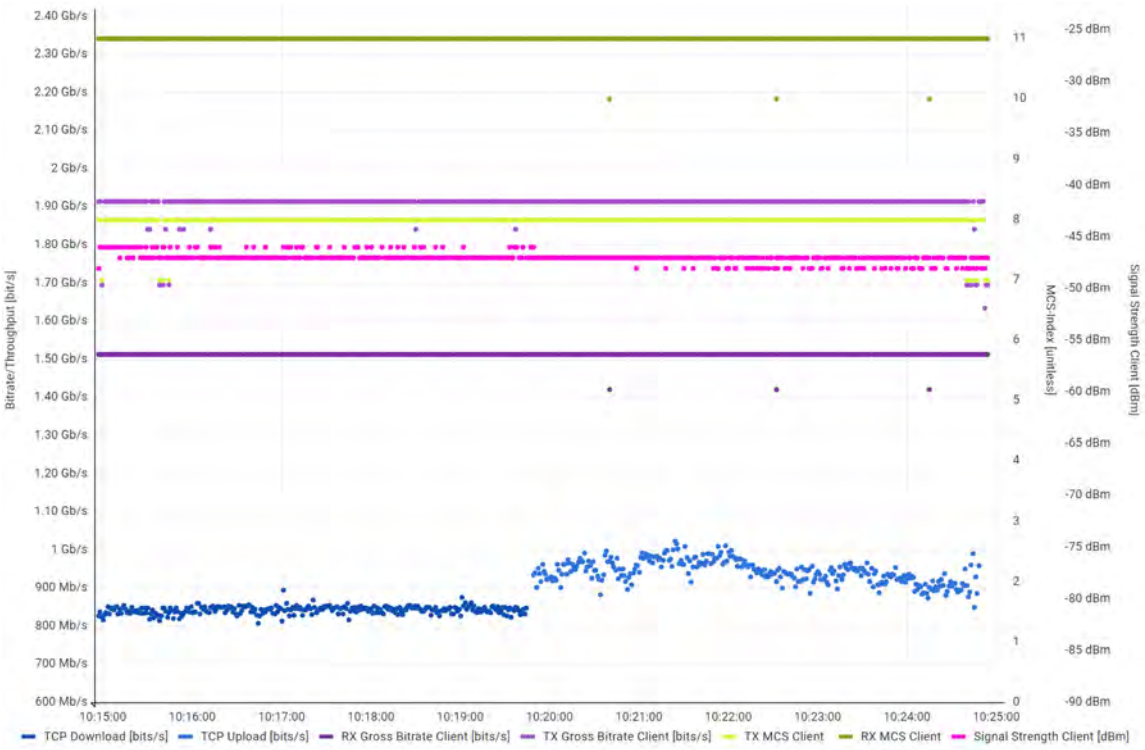

Figure 31: Measurement over 6 GHz with 160 MHz channel width, TCP. Transmit via two spatial streams reaches about 1 Gbit/s. Gross bitrates are wrongly reported here by netlink, RX MCS 11 corresponds to 1201 Mbit/s instead of 1500 Mbit/s and TX MCS 8 corresponds to 1729 Mbit/s

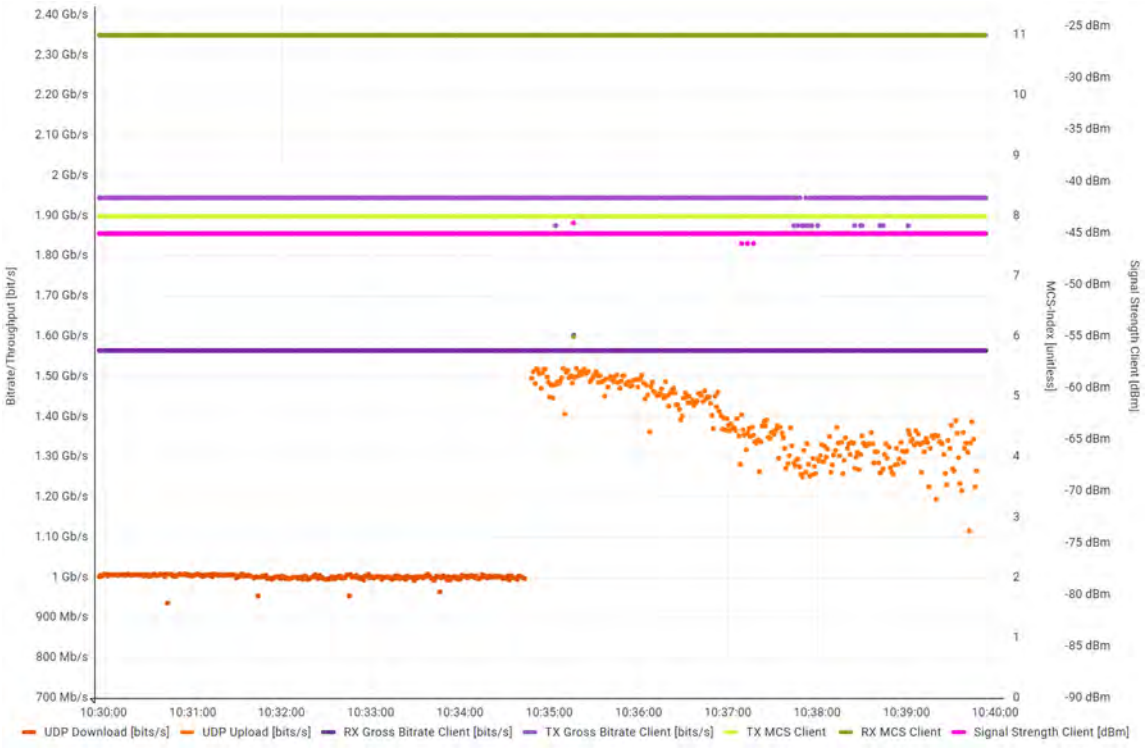

Figure 32: Measurement over 6 GHz with 160 MHz channel width, UDP. Transmit throughput reaches over 1.5 Gbit/s. Gross bitrates are wrongly reported here by netlink, RX MCS 11 corresponds to 1201 Mbit/s instead of 1580 Mbit/s and TX MCS 8 corresponds to 1729 Mbit/s

The diagrams also show an error of the driver: In some cases it calculates the wrong gross data rate from the used MCS when returning it to the measuring program. In the cases shown a data rate of about 1500 Mbit/s, for example is calculated by the driver, which, however, does not exist as a possible data rate, especially not with the negotiated MCS.

If the gross bit rate reported by the driver is disregarded and the achieved throughput is compared with the actual gross bit rate that can be calculated from the MCS, it can be seen that about 83% net throughput is achieved with UDP in RX, for example, and even more than 86% in TX. In comparison, TCP achieves only about 70% (RX) and only about 58% in TX (1000 Mbit/s vs. 1729 Mbit/s). In comparative measurements in the 5 GHz band at least 80% of the gross data rate was also achieved with UPD and about 70% with TCP. By extrapolation, the maximum MCS 11 with two spatial streams can be expected to achieve about 1.9 Gbit/s throughput for UDP under good conditions and about 1.7 Gbit/s for TCP. Under very good conditions, more than 2 Gbit/s may also be possible for UDP. If hardware that reliably supports this MCS is available in the future, this assumption can be verified.

6.2.5 Measurement scenario: Client-to-AP-to-AP-to-Client

Another measurement scenario considered is communication between two clients connected to different APs of the same Extended Service Set: Here, the communication runs over two different channels so that there is no mutual interference in the data transmission (e.g., by selecting channels 15 and 47 in the 6 GHz band, each 160 MHz wide). The APs communicate via the wired connection.

Similar, slightly reduced data throughputs can be expected here at 6 GHz as in the receive in chapter 6.2.4, both in RX and TX direction, since one of the two clients is limited by the missing second spatial stream. In addition, a somewhat stronger scatter of the measurement data is to be expected due to the participation of two air interfaces. Only the last assumption is confirmed in the measurement data: The throughput at 6 GHz falls significantly short of the expected 70%-80% of the negotiated gross bit rate:

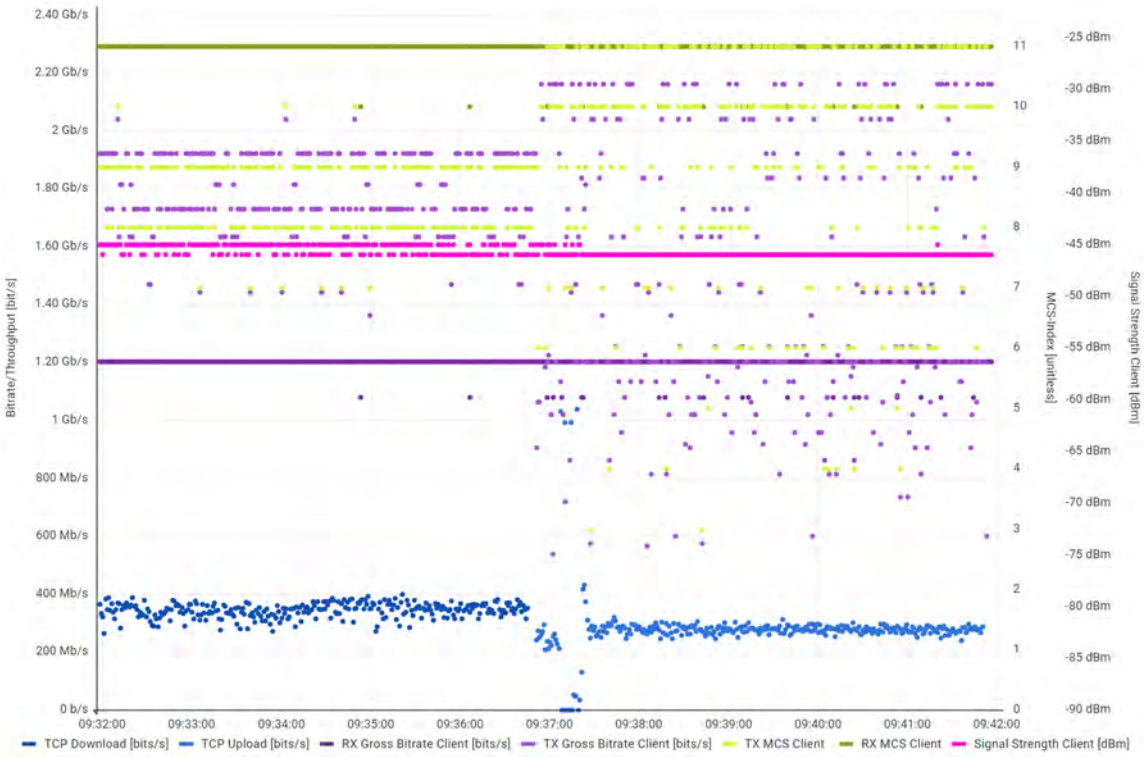

Figure 33: Measurement over 6 GHz with 160 MHz channel width, TCP, between two Clients in two different BSS of an ESS. There are obvious fluctuations in the device transmit bitrates. RX and TX throughput are very low with about 300-400 Mbit/s compared to the expected 1 Gbit/s.

This behavior could be reproduced several times at 6 GHz. A cause for this is not apparent. The picture is different at 5 GHz: The expected throughputs are

achieved here, only the slight reduction of the throughput and stronger scattering compared to a measurement against an external client occurs:

Figure 34: Measurement over 5 GHz with 80 MHz channel width, UDP, between two Clients in two different BSS of an ESS. Throughput reaches 850 Mbit/s which is reduced compared to the 900-1000 Mbit/s reached with an external measurement server.

6.2.6 Measurement scenario: AP-Handover/Roaming

In advance of this measurement the transmission power of the two access points used was reduced so that the roaming range in the 6 GHz band was approximately in the middle of the corridor between the two rooms used.

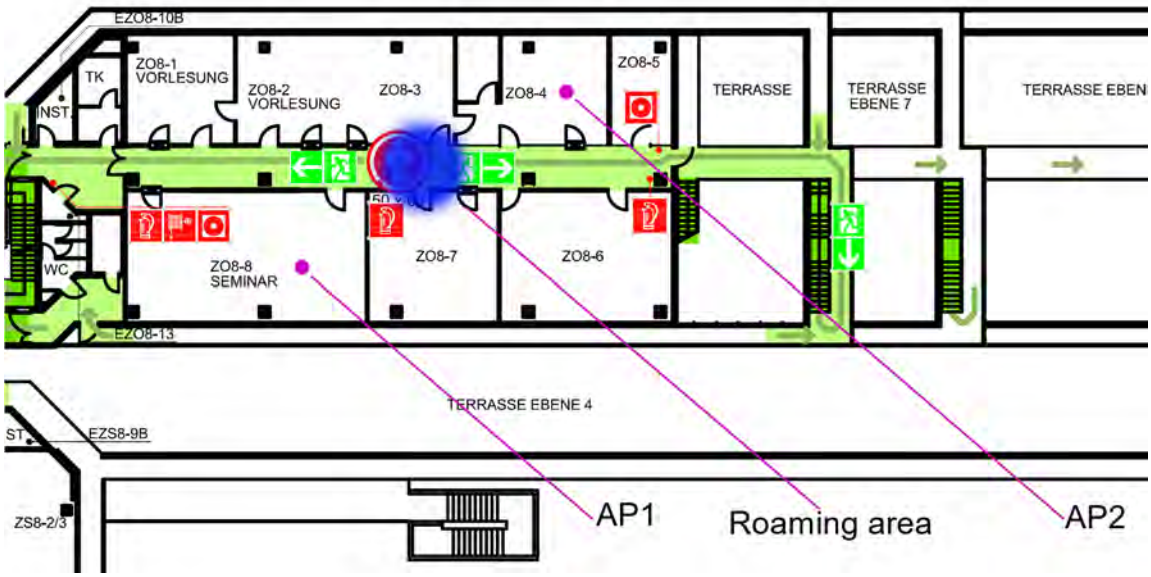

Figure 35: The signal strength of the APs was measured and an ideal roaming area was defined in the hallway between the two rooms. AP transmit power was reduced to fit this area.

Clients usually have a threshold at which the signal strength of the previous associated access point is so low that they switch to the other access point. Typically, a client switches either immediately or shortly after the signal strength in the other cell is greater than in the current cell [59]. Depending on the manufacturer the hardware behaves differently (different limits) and parameters such as roaming aggressiveness or support for Neighbor Reports with 802.11k also change the roaming behavior. In addition, some clients are not able to use roaming at all.

For the AX210, the tests show the following picture in this respect: Under Windows 11 with driver 22.130.0.5 the NIC is able to perform roaming between the two APs. The roaming process interrupts the data transmission for about one second. Under Windows the roaming aggressiveness can be configured as a parameter in the driver settings. We tested with the value set to "medium". This parameter controls the reception power at which the NIC automatically scans for other access point candidates in the environment [60].

However, if the AX210 is used under Linux 5.17 with driver iwlwifi-ty-a0-gf-a0-69 the NIC behaves like a sticky client: Roaming in 6 GHz does not work then. Even if the transmission power of the APs is reduced to 10 dBm (resulting in between -80 dBm and -85 dBm to the remote AP on the client side) roaming to the other access point is not possible. Only when the client completely loses the connection (i.e. moves out of the reception range of the old AP), a scan is performed and then the client associates with the nearest AP. Roaming could not be configured via the driver interface. It is not clear here whether roaming is perhaps disabled on the driver side or possibly also not possible with Linux drivers so far. A bug report opened for this did not bring an answer from Intel yet [61]. It should be noted that this NIC can also be used explicitly in mobile devices like laptops due to its M.2 form factor. M.2 is the de-facto standard for integrating WLAN and Bluetooth in these mobile devices. In such cases the lack of roaming properties is a problem.

6.2.7 Measurement scenario: Distance Measurements

The distance measurements were performed with the minimum channel bandwidth, since the primary limitation here is the distance to an access point. Measurements were first performed directly under the access point (about 2 meters away from the AP) and then with increasing distance (2-meter increments) to the AP. Upload and download were measured for 30 seconds each. Only TCP was tested, since a sufficient connection to the AP can be assumed if a connection is established with TCP (handshake) and data is then successfully transferred. Both the 6 GHz band and the 5 GHz band were tested to enable a comparison. The same transmission power (abbreviated as TP in the following) was selected for the AP in the 5 GHz as in the 6 GHz: 23 dBm. Due to the spatial conditions, there is a sharp drop in the signal strength of the AP between the 8-meter measurement and the 10-meter measurement. This is due to the double doors located there (see figure below). The concrete columns in the corridor also cause fluctuations in the signal strength.

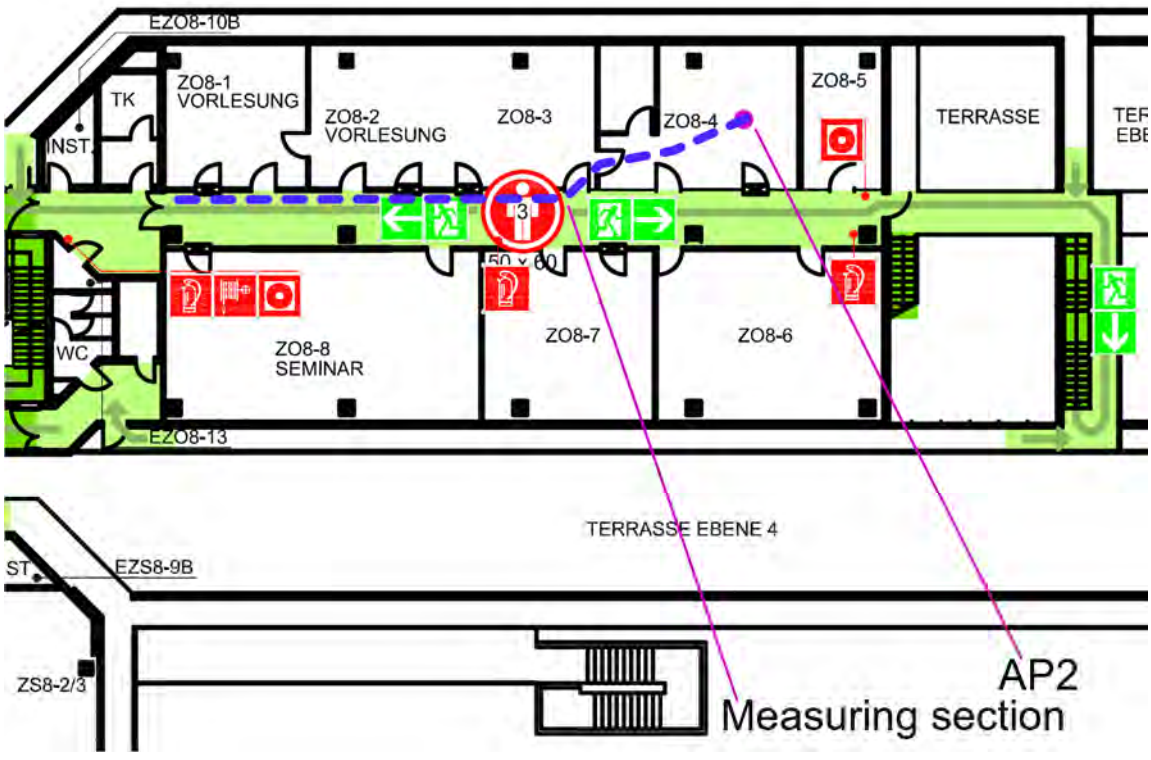

Figure 36: Distance measurement section where the clients was moved along from the AP further into the hallway to the entrance doors.

The total measurement distance is 26 meters. This is not sufficient to determine the maximum reception distance, since sufficiently stable reception is still possible

at 26 meters: For example, an MCS index of 7 was still possible at 26 meters in the 6 GHz band, i.e. a gross bit rate of 86 Mbps. The power of the access point then had to be reduced further in order to measure the distance again. The distance limit was then reached at (extrapolated) 40 meters for both 5 GHz and 6 GHz. After this distance, i.e. 42 meters, a successful TCP connection was no longer possible in either case.

Also to be observed were the fluctuations in the achieved data throughput, which occur more frequently with increasing distance; short-term drops in the bit rate as well as dropouts in the data transmission also increasingly occur here.

The following are the measurements with 6 GHz band, first with 23 dBm transmit power of the AP, then with reduced transmit power. The measurements with reduced transmit power were taken from the point where the reception quality is comparable to the end point of the last series of measurements: With transmit power reduced to 9 dBm, the 10-meter measurement point is approximately comparable to the 26-meter measurement point of the 23 dBm measurement. A perfect comparability is not given, since some parameters cannot be influenced, e.g. software logic for the selection of MCS and spatial streams. Just like the transmission power of the AP the client-side transmission power was also reduced to 9 dBm in order to create as much comparability as possible in the upload. The only notable difference between 5 GHz and 6 GHz is the different selection of spatial streams and the associated MCS indices described above. It should be noted, however, that at 5 GHz a higher transmission power of the AP (> 23dBm) can be selected in practice, since up to 1 Watt can be transmitted there for some channels. This means that in practice a higher range can be achieved than at 6 GHz, where the limitation of the transmitting power by the BNetzA has an effect.

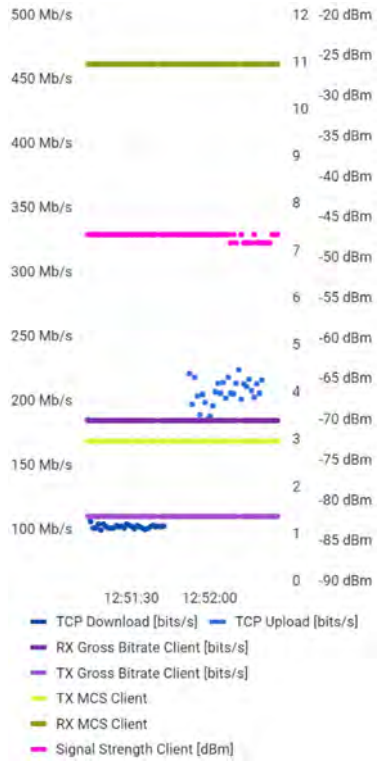

Figure 37: 6 GHz 20 MHz client at 2 m, 23 dBm TP

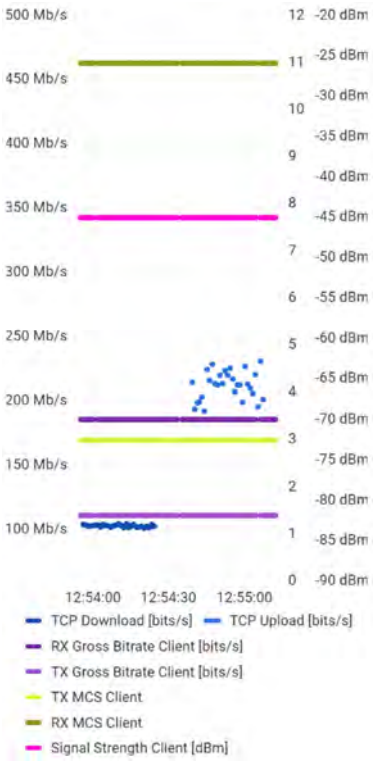

Figure 38: 6 GHz 20 MHz client at 4 m, 23 dBm TP

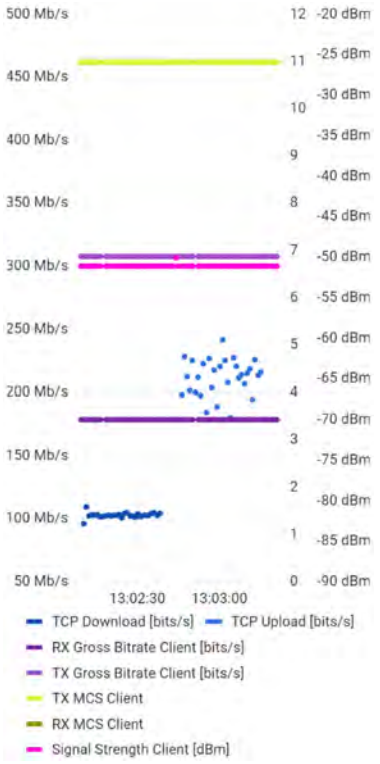

Figure 39: 6 GHz 20 MHz client at 6 m, 23 dBm TP

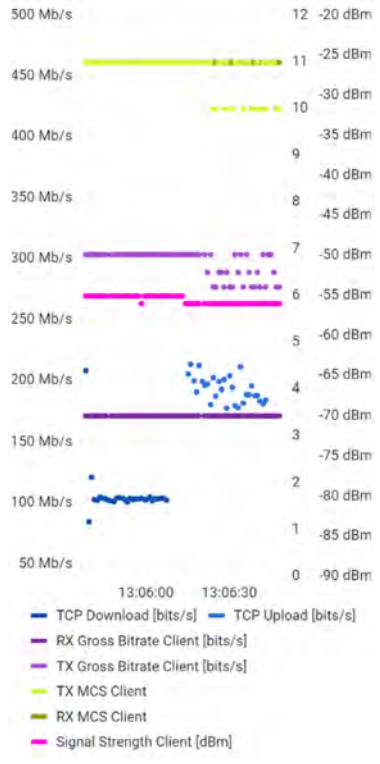

Figure 40: 6 GHz 20 MHz client at 8 m, 23 dBm TP

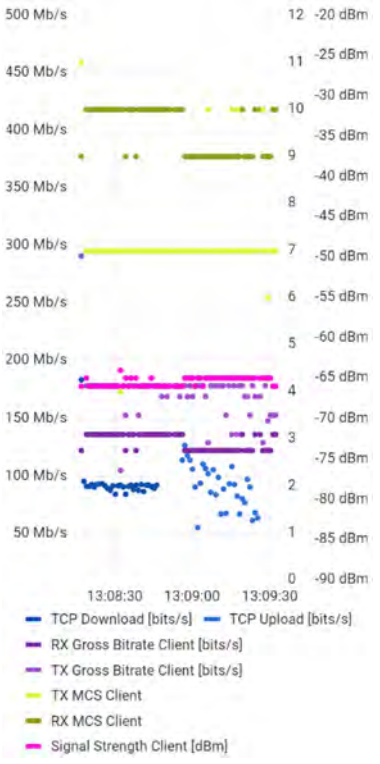

Figure 41: 6 GHz 20 MHz client at 10 m, 23 dBm TP

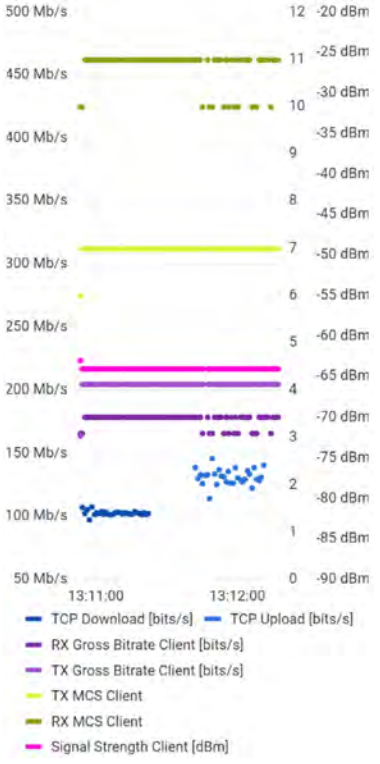

Figure 42: 6 GHz 20 MHz client at 12 m, 23 dBm TP

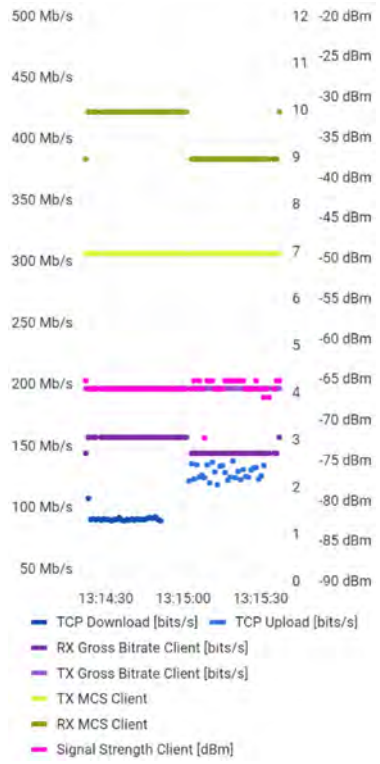

Figure 43: 6 GHz 20 MHz client at 14 m, 23 dBm TP

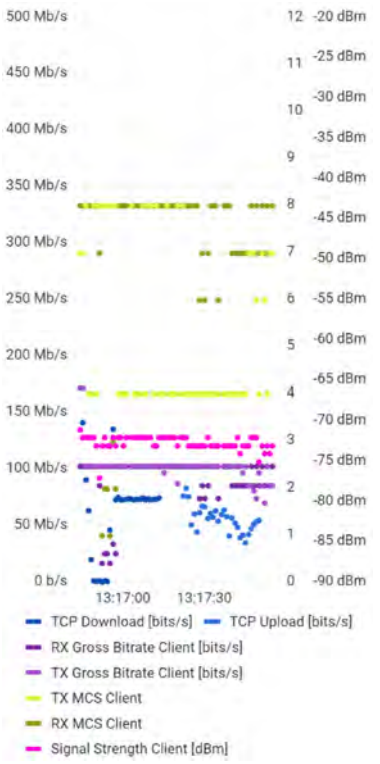

Figure 44: 6 GHz 20 MHz client at 16 m, 23 dBm TP

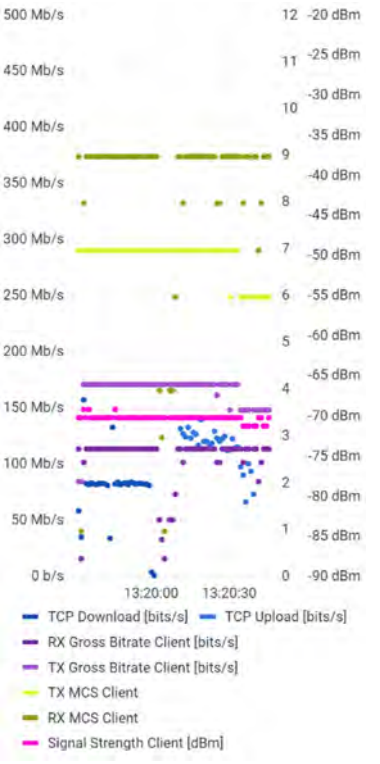

Figure 45: 6 GHz 20 MHz client at 18 m, 23 dBm TP

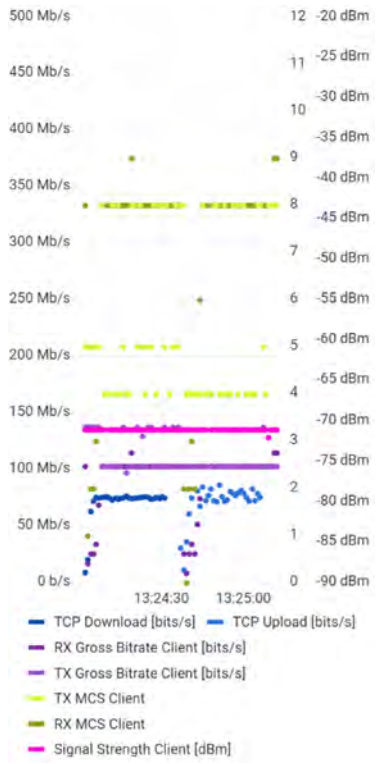

Figure 46: 6 GHz 20 MHz client at 20 m, 23 dBm TP

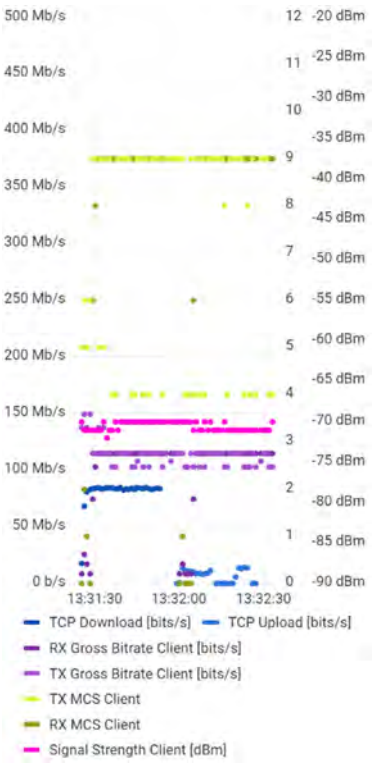

Figure 47: 6 GHz 20 MHz client at 22 m, 23 dBm TP

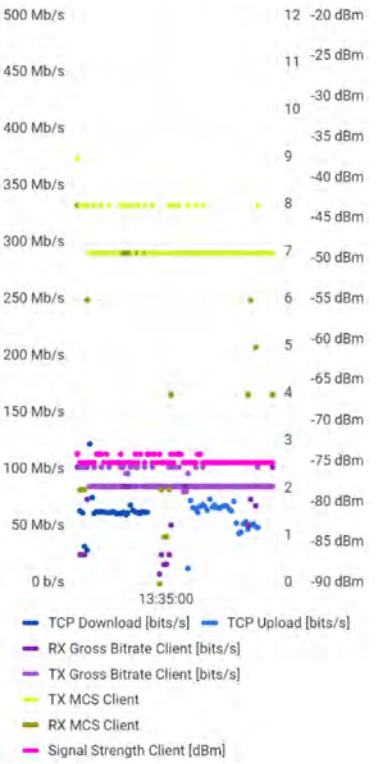

Figure 48: 6 GHz 20 MHz client at 24 m, 23 dBm TP

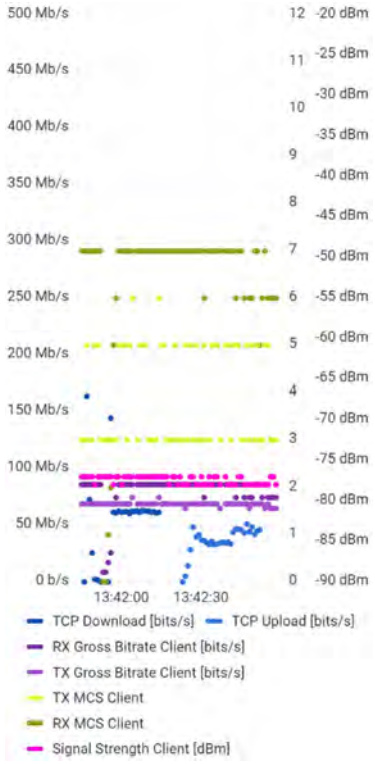

Figure 49: 6 GHz 20 MHz client at 26 m, 23 dBm TP

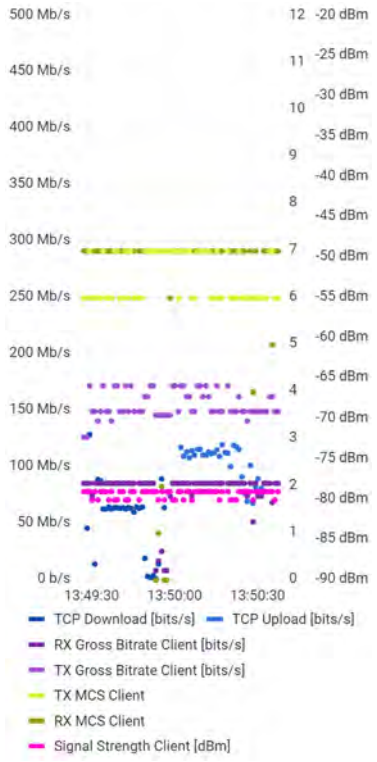

Figure 50: 6 GHz 20 MHz client at 10 [26] m, 9 dBm TP

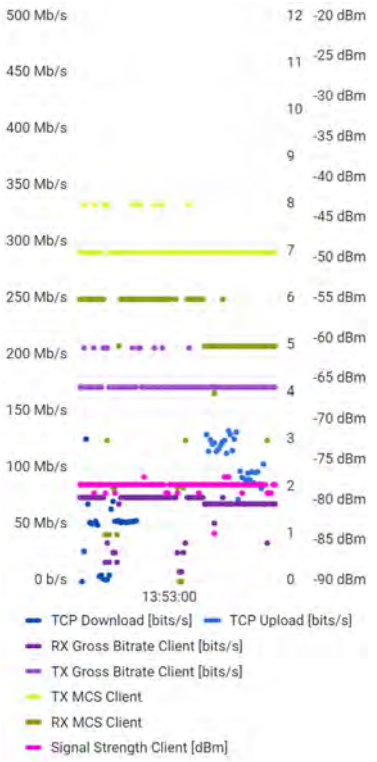

Figure 51: 6 GHz 20 MHz client at 12 [28] m, 9 dBm TP

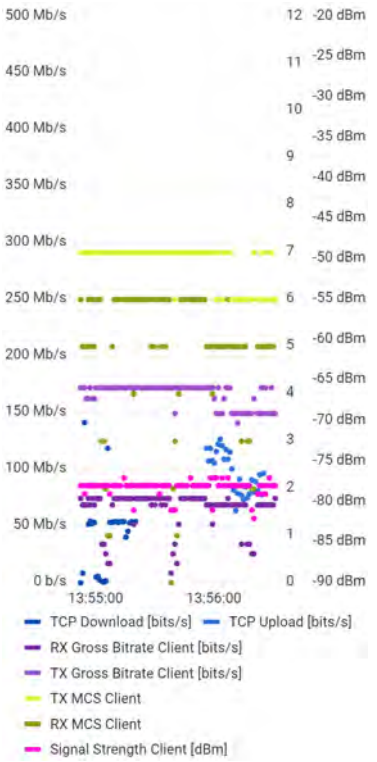

Figure 52: 6 GHz 20 MHz client at 14 [30] m, 9 dBm TP

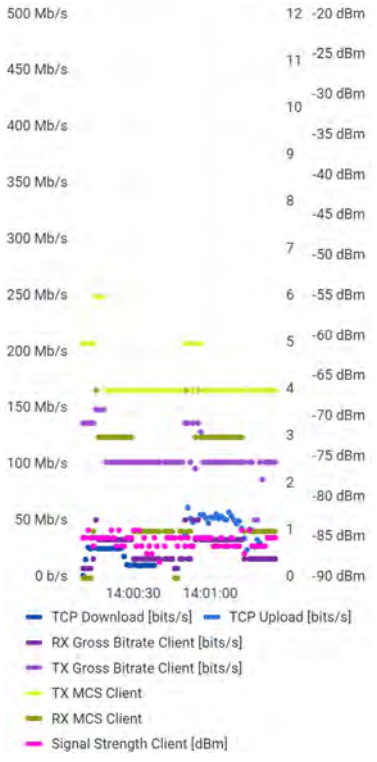

Figure 53: 6 GHz 20 MHz client at 16 [32] m, 9 dBm TP

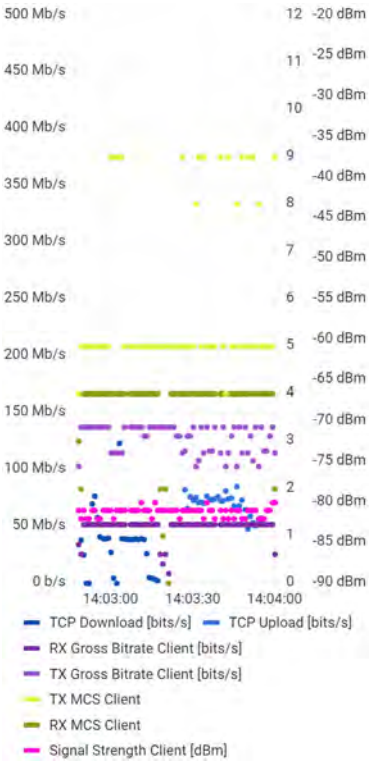

Figure 54: 6 GHz 20 MHz client at 18 [34] m, 9 dBm TP

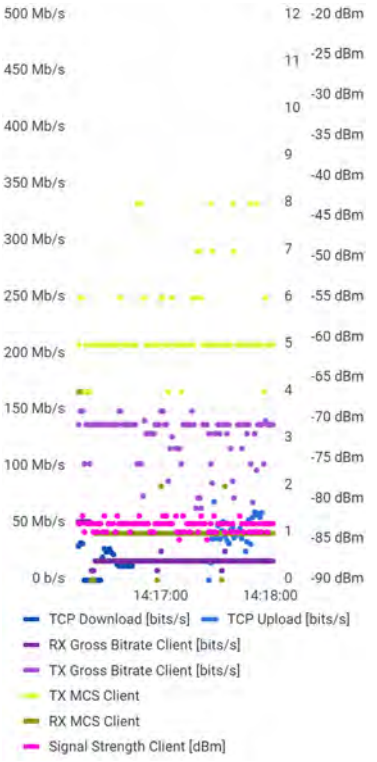

Figure 55: 6 GHz 20 MHz client at 20 [36] m, 9 dBm TP

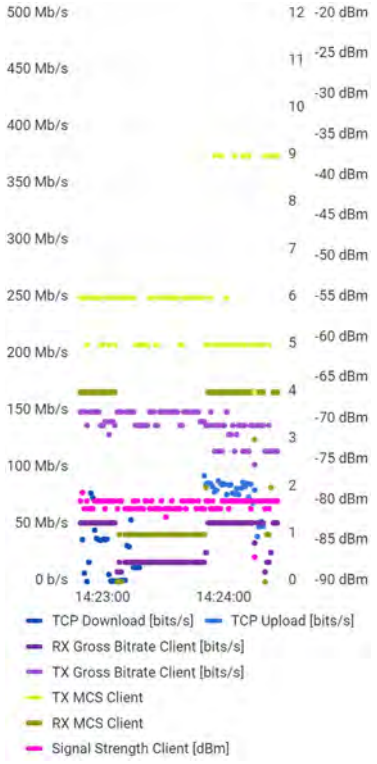

Figure 56: 6 GHz 20 MHz client at 22 [38] m, 9 dBm TP

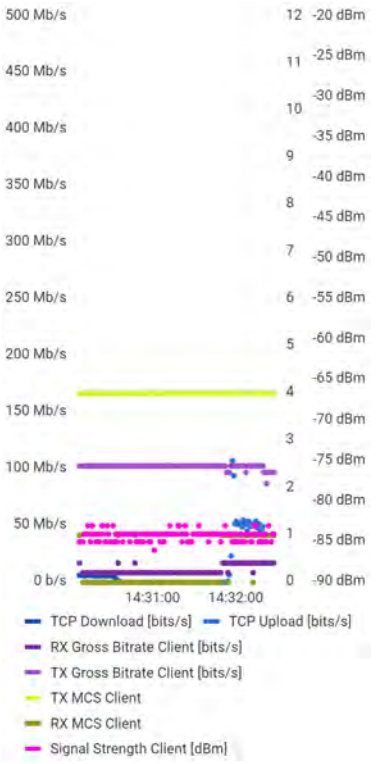

Figure 57: 6 GHz 20 MHz client at 24 [40] m, 9 dBm TP

6.2.8 Regarding OFDMA

Originally, each of the tests carried out above was also to be tested with OFDM in addition to OFDMA, i.e. modulation, as part of this work. However, OFDMA is not used directly by all subscribers in a network who support it, but is negotiated between individual (or all) subscribers and the access point. It can also be activated individually for the uplink or downlink [62]. It allows simultaneous transmission of different subscribers within one OFDM symbol by dividing the subcarriers among the participants [63]. This allocation of resource units (RUs) is done via trigger frames of which there are several types: Basic trigger frames, multi-user request-to-send (MU-RTS) frames, buffer status report frames, bandwidth query report poll (BQRP) and several more. The access point informs the participating STAs that they can use OFDMA with a certain amount of RUs.

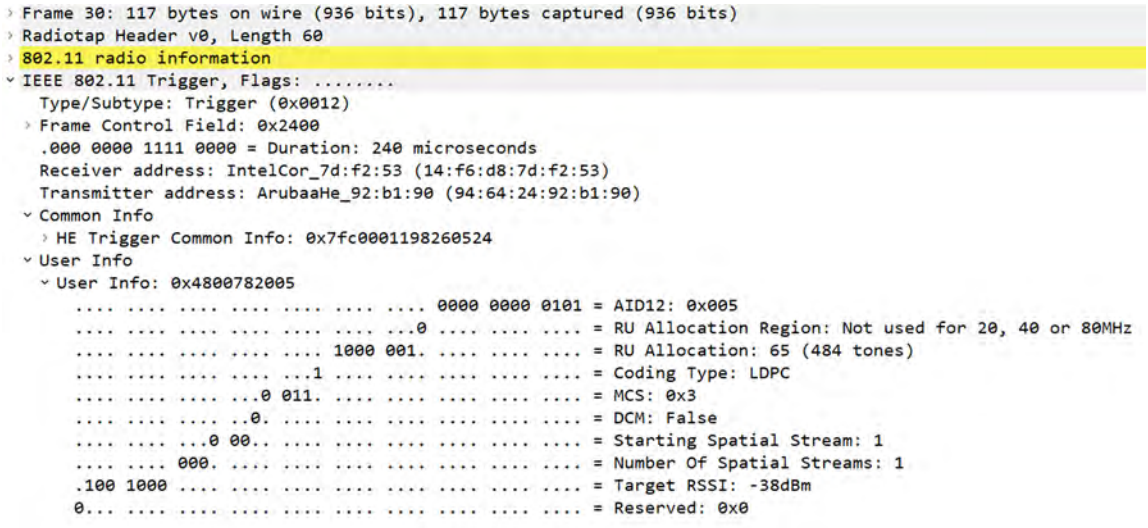

Figure 58: Trigger Buffer Status Report Poll (BSRP) Frame (a Trigger frame), sent from the Aruba AP to an Intel NIC telling it to use 484 tones of the 80 MHz channel, which is half of it.

Regardless of whether UL-OFDMA or DL-OFDMA is to be used the access point must allocate the RUs to the STAs. In practice, the Aruba APs used also sent the necessary trigger frames. Wireshark captures for this are linked in the appendix for download. If OFDMA is used in the download an STA must respond with a clear-to-send after it has received an MU-RTS frame as a trigger, thereby confirming that the STA will use OFDMA in the future. The AP then sends multi-user DL PPDUs to the STAs and requests acknowledgement of receipt with a block ACK request (BAR). The STAs then each respond with their own Block ACK.

OFDMA was tested with the AX210 in the 6 GHz band as well as in the 5 GHz band (802.11ax). An Intel AX201 (under Windows as well as MacOS) and an Apple iPhone with 802.11ax support were then also tested in the 5 GHz band. The AX210 was tested under Linux as well as Windows. No transmission with OFDMA could be reliably determined for any of the cards mentioned. The STAs partially responded to the trigger frames, e.g., an AX210 could be observed sending a block ACK, but only to a trigger frame that allocates the entire width (i.e. all subcarriers/tones) to the client, which corresponds to operation with OFDM:

ArubaaHe_92:b1:90	Broadcast	802.11	433	Beacon frame, SN=1702, FN=0, Flags=........C, BI=100, SSID=LAB5
ArubaaHe_92:b1:90 …	IntelCor_bc:c7:44 …	802.11	117	Trigger Buffer Status Report Poll (BSRP), Flags=........
IntelCor_bc:c7:44 …	ArubaaHe_92:b1:90 …	802.11	76	Request-to-send, Flags=........C
IntelCor_bc:c7:44	Broadcast	802.11	140	QoS Data, SN=46, FN=0, Flags=.p.....TC
Dell_ef:b2:e7	IntelCor_bc:c7:44	802.11	154	QoS Data, SN=8, FN=0, Flags=.p....F.C
IntelCor_bc:c7:44 …	ArubaaHe_92:b1:90 …	802.11	92	802.11 Block Ack, Flags=........C
ArubaaHe_92:b1:90	Broadcast	802.11	433	Beacon frame, SN=1703, FN=0, Flags=........C, BI=100, SSID=LAB5

Figure 59: Intel NIC acknowledging a BSRP Trigger frame which allocated all channel subcarriers to the NIC, not a subset.

In principle, after OFDMA has been successfully negotiated between the AP and one or more STAs, data frames should be transmitted via the respective allocated resource units. For the measuring station that listens to OFDM-modulated frames on the channel (with which the captures were made) this means that OFDMA-modulated data frames are not recorded in the capture: Data frames that use, for example, a 484-tone RU at 80 MHz channel bandwidth are not captured. However, data frames from clients that have not actually been allocated the entire channel bandwidth are still captured in the captures: So, despite being told to use certain RUs, these clients continue to use the entire channel. In addition, when observing the channel occupancy (spectrum analysis), we were always able to detect utilization on the entire channel bandwidth. This both indicates that none of the clients in use is currently using OFDMA successfully. However, OFDMA-modulated communication could possibly be observed in another case with a Samsung S10e and another access point Cisco (Catalyst) 9115 AP. But there is no certainty here either: In particular, OFDMA could not be reliably and reproducibly negotiated here either, but only in one test case [64]. Further, more in-depth analysis is therefore necessary here. Of particular interest here would be the possibility of recording OFDMA-modulated frames in Wireshark as well as a more detailed analysis of the frequency spectrum (i.e. recording of the subcarriers and their utilization).

Our observation at least shows the correct coordination of the allocation of RUs by the trigger frames of the AP. Only the other participating STAs do not yet react to this in such a way that OFDMA is used. However, the Wi-Fi Alliance has certified the NICs used for Wi-Fi 6 and Wi-Fi 6E [65] [66]. The certification also clearly states the support of OFDMA as well as the support of trigger frames. Upon request to the Wi-Fi Alliance, it has not yet been possible to determine how the Wi-Fi Alliance could successfully test OFDMA [67].

6.3 Reference values in the 5 GHz Frequency Band

In order to be able to make a better comparison between the measurements in the 6 GHz band and the previously possible 5 GHz band, measurements were also made with 80 MHz channel bandwidth in both bands. 160 MHz was not possible in the 5 GHz band, so no comparison can be made for the maximum possible channel bandwidth in the 6 GHz band.

Figure 60: Measurement over 6 GHz with 80 MHz channel width, UDP. Due to only one spatial stream in the download direction download throughput is not directly comparable to download throughput with 5 GHz, which uses two spatial streams.

In terms of throughput, the measurements with 802.11ax in the 5 GHz band and 6 GHz band are almost identical if you disregard the fact that a spatial stream is missing in the download at 6 GHz. The maximum MCS index 11 is even selected here at 80 MHz in both cases. At 6 GHz, however, only for receive (i.e. only with a spatial stream) and at 5 GHz the high bit rate cannot be maintained as soon as a data transmission occurs: Here the MCS index is reduced to 9. Also worth mentioning here is the slight drop in the data transmission rate in transmit at 6 GHz via UDP. This behavior also occurs at 160 MHz channel bandwidth (see above), but is not found at 5 GHz. An explanation for the phenomenon could not be found. TCP does not show this behavior.

Figure 61: Measurement over 5 GHz with 80 MHz channel width and 802.11ax enabled, UDP. Throughput is like 6 GHz upload in both directions, due to using two spatial streams. Note the change in the selected MCS/bitrate when actually transmitting or receiving and it changing when the direction is not in use.

As an addition, the same measurement with 802.11ac is shown here. Note that the OFDM uses a different subcarrier spacing and OFDM symbol duration for the same MCS index, so the gross data rate is different:

Figure 62: Measurement over 5 GHz with 80 MHz channel width and 802.11ac. Bitrate is lower despite same MCS as in the figure above due to different OFDM characteristics.

6.4 NetworkManager Problems on Debian

The measurements were performed under Debian Sid with kernel 5.17. However, some measurements were not usable because they showed the following phenomenon:

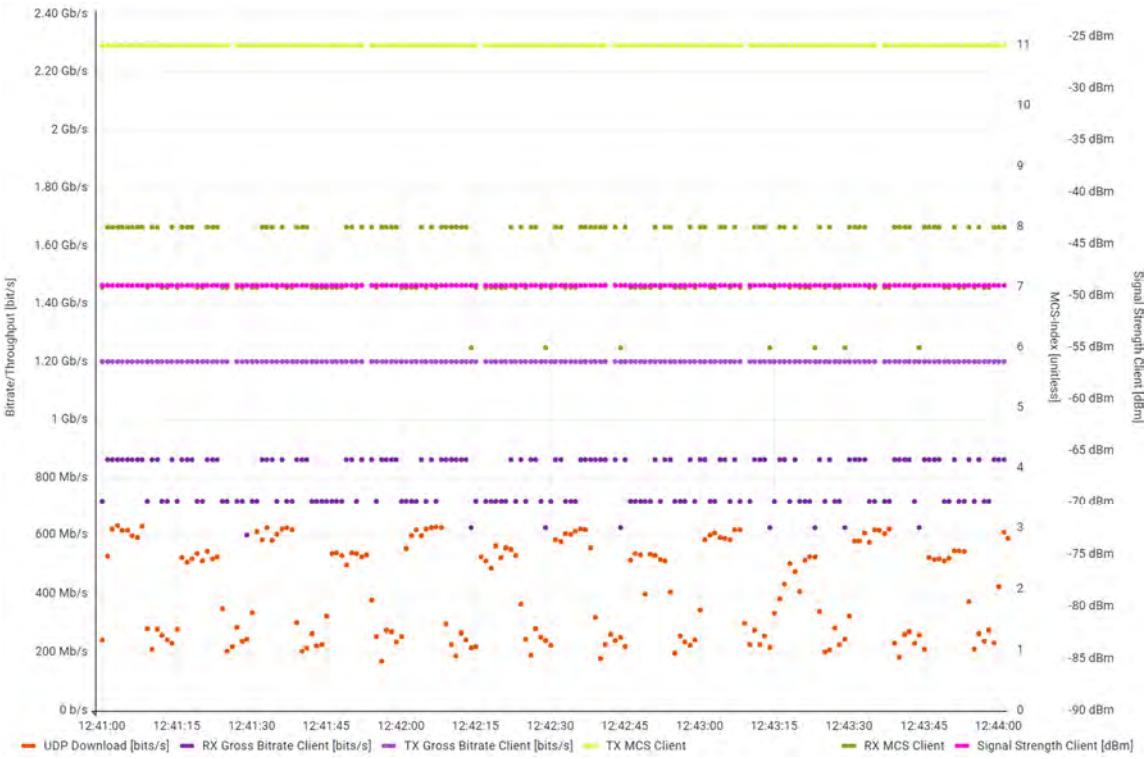

Figure 63: Debian NetworkManager causes low throughput while scanning on the interface for seven second intervals.

About every fourteen seconds the data throughput drops by about 60% for seven seconds and then recovers. This regular pattern could be observed at both 5 GHz and 6 GHz.

Figure 64: Debian NetworkManager scans for a network and while doing so, reduces throughput on the interface.

The cause is the NetworkManager component of Debian: When the GUI interface of the NetworkManager is open (via this you see a list with the available WLAN

networks) an active scan for available networks is performed every seven seconds to update this list. Exactly during this scan interval, which lasts seven seconds, the drop in data throughput occurs. These scans should actually be executed in the background and not affect the throughput. The behavior could be circumvented in the further tests by closing the GUI component before each test. This meant that the problem did not occur any longer.

7 Summary and Future Prospects

The automated management of Wi-Fi 6E networks and IoT devices in these networks can be well mapped with ThingsBoard. New devices can be integrated into the management platform via a provisioning process and also configured centrally. Communication between the devices and the platform can be authenticated and confidential. The devices can be controlled and (measurement) data can be received and sent. Central monitoring is possible as well as diagnostic collection of audit logs of the platform or device logs. Updates can also be managed centrally and applied to the devices. The flexible rule chains allow granular logic with which, for example, data can be processed, or the control of the devices can be carried out. Thinger.io takes a slightly different approach with the primary target group of microcontrollers, not the Linux systems used here in this thesis. In addition, some points, such as the control of the devices or the collection of data, are not as individually configurable or offer a smaller range of functions.

In the future we should take a closer look at the behavior of ThingsBoard in a large-scale deployment: Only a few devices with Wi-Fi 6E support were available here so further questions arise when ThingsBoard works with many simultaneously managed devices: How do rule chains behave in such cases, does this affect performance in terms of telemetry data processing or provisioning? Are there bandwidth issues when rolling out firmware updates? Such a setup could perhaps also be realized more closely by a large number of simulated or virtualized devices to be managed.

The focus of Thinger.io on microcontrollers urges a renewed evaluation of the software with, for example, Arduino devices, in order to be able to take a closer look at the differences to the use with the not yet fully developed HTTP API. 802.11ax-compatible microcontroller boards could then be used, for instance, to evaluate Wi-Fi 6 or Wi-Fi 6E.

Regarding the Quality of Service in the 6 GHz band (Wi-Fi 6E), it can be seen that the net data throughput lags behind the gross bit rate: Only between 58%-86% of the gross bit rate is actually achieved as UDP or TCP data throughput, with TCP expectedly slightly lower. In the tests the maximum MCS with two spatial streams could never be achieved for the optimal configuration (6 GHz channel with 160

MHz channel bandwidth). This was only possible with 80 MHz. Throughputs above 1 Gbit/s were nevertheless measured with both TCP and UDP. If the values measured here are taken as a basis, up to 2 Gbit/s can be expected for the maximum MCS index 11 with two spatial streams for UDP and about 1.7 Gbit/s for TCP. This could not be verified within the scope of this work but should be achievable in the future (perhaps through newly available drivers or other hardware). New drivers are also necessary for this reason alone, in order to address the existing errors: Currently, two spatial streams are not possible in the download in the 6 GHz band, nor could the maximum MCS be negotiated under Linux, despite good channel characteristics. Hardware with support for four spatial streams can also be expected in the foreseeable future. The use of such NICs is rather unlikely in the IoT environment but offers maximum throughput for other use cases.

Some parameters of the Intel AX210 cannot yet be configured or accessed via the currently used driver, e.g. the modulation cannot yet be actively influenced, to for example prefer OFDMA on the client side. In general, OFDMA is correctly supported by the access points used and the APs send out trigger frames to allocate the resource units to the subscribers, but none of the devices tested currently supports OFDMA sufficiently to use this allocation. Data transfer via OFDMA cannot be observed. We can also hope for newer client hardware or drivers, especially from other manufacturers, to be able to analyze differences in the use of OFDMA.

Under Linux the AX210s were also unable to perform correct roaming between two APs; this was only possible under Windows 11. In the future an evaluation of the test cases carried out under Windows could be considered, provided that the measurement software can be ported - here, the control of the NIC via the driver is particularly decisive: Under Windows some settings can be changed in the driver options, which are not possible under Linux (e.g., the roaming aggressiveness).

Due to the limited amount of hardware so far and the fact that Wi-Fi 6E has not yet reached an advanced stage of deployment. Tt is not yet possible to make any statements on other points either: How will the restrictions on transmission power imposed by the Bundesnetzagentur affect deployments in Germany in reality? Will there be differences in the rollout between different states because of this?

A mesh topology in the 6 GHz band was also not considered in detail, although it could also be useful in the IoT environment. Here, however, it is to be hoped for better availability of Wi-Fi 6E-capable hardware that will enable the implementation of a mesh network and other future testing of more complex environment settings. Likewise, the 802.11ax standard results in further points such as BSS coloring and beamforming, which were not considered in this work, but could also have an influence with regard to data throughput.

References

[1] S. Forshee, "Central Regulatory Domain Agent," 23 03 2015. [Online]. Available: https://wireless.wiki.kernel.org/en/developers/regulatory/crda. [Accessed 12 09 2019].

[2] IEEE Standards Association, LAN/MAN Standards Committee, "IEEE Standard 802 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," The Institute of Electrical and Electronics Engineers, New York, 2020.

[3] Wi-Fi Alliance, "Member Companies," 2022. [Online]. Available: https://www.wi-fi.org/membership/member-companies. [Accessed 18 03 2022].

[4] Wi-Fi Alliance, "Wi-FI Alliance Generational Naming," 2022. [Online]. Available: https://www.wi-fi.org/sites/default/files/public/images/Generational_naming_20210602.png. [Accessed 18 03 2022].

[5] IEEE Standards Association, LAN/MAN Standards Committee, "4.3.15a High-efficency (HE) STA," in *IEEE Standard 802 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendmend 1: Enhancements for High-Efficency WLAN*, New York, The Institute of Electrical and Electronics Engineers, 2021, p. 47.

[6] European Telecommunications Standards Institute, "System Reference document (SRdoc); Wireless access systems including radio local area networks (WAS/RLANs) in the band 5 925 MHz to 6 725 MHz," ETSI, Sophia Antipolis Cedex, FRANCE, 2018.

[7] Bundesnetzagentur, "Vfg. 55/2021," 30 06 2021. [Online]. Available: https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Allgemeinzuteilungen/MobilfunkDectWlanCBFunk/vfg552021WLAN6GHz.pdf?__blob=publicationFile&v=3. [Accessed 04 04 2022].

[8] Bundesnetzagentur, "Vfg. 151/2018," 2018. [Online]. Available: https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Allgemeines/Presse/Pressemitteilungen/AmtsblattVfgId334pdf.pdf?__blob=publicationFile&v=3. [Accessed 04 04 2022].

[9] Oracle, "What is IoT?," 2022. [Online]. Available: https://www.oracle.com/internet-of-things/what-is-iot/. [Accessed 13 06 2022].

[10] Realtek Semiconductor Corp., "RTL8156B(S)-CG - REALTEK," 2019. [Online]. Available: https://www.realtek.com/en/products/communications-network-ics/item/rtl8156b-s-cg. [Accessed 18 04 2022].

[11] magellan netzwerke GmbH, "magellan – Ihr Full Service Security Spezialist | magellan netzwerke GmbH," 04 2022. [Online]. Available: https://www.magellan-net.de/de/. [Accessed 18 04 2022].

[12] Intel Corporation, "Intel® Active Management Technology (Intel® AMT) | Intel," 10 01 2021. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-active-management-technology.html. [Accessed 28 04 2022].

[13] Intel Corporation, "Intel WiFi 6E AX210 Gig Produktspezifikationen," 2020. [Online]. Available: https://ark.intel.com/content/www/de/de/ark/products/204836/intel-wifi-6e-ax210-gig.html. [Accessed 18 04 2022].

[14] P. Kalytta and G. Ben Ami, "215043 – iwlwifi: AX210: Allow 6 GHz Wi-Fi 6E operation in Germany," Kernel.org Bugzilla, 24 02 2022. [Online]. Available: https://bugzilla.kernel.org/show_bug.cgi?id=215043. [Accessed 18 04 2022].

[15] S. Forshee, A. Mohr and X. Vasquez Perez, "en:developers:regulatory:wireless-regdb [Linux Wireless]," Linux Kernel Wiki, 21 07 2021. [Online]. Available: https://wireless.wiki.kernel.org/en/developers/regulatory/wireless-regdb. [Accessed 12 05 2022].

[16] L. Coelho, "205695 – [iwlwifi] 9260AC crashes with lar_disable=1," 16 12 2019. [Online]. Available: https://bugzilla.kernel.org/show_bug.cgi?id=205695#c6. [Accessed 13 06 2022].

[17] Hewlett Packard Enterprise Development LP, "630 Series Wi-Fi 6E Indoor Access Points | Aruba," 04 2022. [Online]. Available: https://www.arubanetworks.com/products/wireless/access-points/indoor-access-points/630-series/. [Accessed 23 04 2022].

[18] Aruba, "Aruba 630 Series Campus Access Points - Datasheet," 08 04 2022. [Online]. Available: https://www.arubanetworks.com/resource/aruba-630-series-access-points-data-sheet/. [Accessed 23 04 2022].

[19] OpenRemote Inc., "OpenRemote | The 100% Open Source IoT Device Management Platform," 2022. [Online]. Available: https://openremote.io. [Accessed 13 04 2022].

[20] INTERNET OF THINGER S.L., "Thinger.io – Open Source IoT Platform," 2020. [Online]. Available: https://thinger.io. [Accessed 13 04 2022].

[21] The Thingsboard Authors, "ThingsBoard - Open-source IoT Platform," 2022. [Online]. Available: https://thingsboard.io. [Accessed 13 04 2022].

[22] Mainflux Labs, " Mainflux Open Source IoT Platform," 2020. [Online]. Available: https://mainflux.com. [Accessed 13 04 2022].

[23] The Thingsboard Authors, "GitHub - thingsboard/thingsboard: Open-source IoT Platform - Device management, data collection, processing and visualization.," GitHub,

Inc., 15 04 2022. [Online]. Available: https://github.com/thingsboard/thingsboard. [Accessed 15 04 2022].

[24] The ThingsBoard Authors, "ThingsBoard Documentation | ThingsBoard Community Edition," 2022. [Online]. Available: https://thingsboard.io/docs/. [Accessed 15 04 2022].

[25] INTERNET OF THINGER S.L., "DATA BUCKETS - Thinger.io Documentation," 04 2021. [Online]. Available: https://docs.thinger.io/features/buckets. [Accessed 15 04 2022].

[26] Hewlett Packard Enterprise, "VisioCafe free visio stencils download site," VSD Grafx Inc., 04 04 2022. [Online]. Available: http://www.visiocafe.com/hpe.htm. [Accessed 19 04 2022].

[27] The Thingsboard Authors, "Entities and relations | ThingsBoard Community Edition," 2022. [Online]. Available: https://thingsboard.io/docs/user-guide/entities-and-relations/. [Accessed 12 05 2022].

[28] INTERNET OF THINGER SL, "OVERVIEW - Thinger.io Documentation," 09 2021. [Online]. Available: https://docs.thinger.io/#thinger.io-main-features. [Accessed 23 04 2022].

[29] INTERNET OF THINGER SL, "HTTP Plugin Documentation," 2021. [Online]. Available: https://docs.thinger.io/plugins/http. [Accessed 23 04 2022].

[30] D. Coleman, "Subcarriers - TIP," in *Wi-Fi 6 & 6E for dummies*, Hoboken, New Jersey, John Wiley & Sons, Inc., 2022, p. 20.

[31] The iperf Authors, ESnet, "GitHub - esnet/iperf: iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool," 18 04 2022. [Online]. Available: https://github.com/esnet/iperf. [Accessed 28 04 2022].

[32] K. Yan, "ping3 · PyPI," 19 04 2022. [Online]. Available: https://pypi.org/project/ping3/. [Accessed 28 04 2022].

[33] The ThingsBoard Authors, "Python REST Client | ThingsBoard Community Edition," 10 2021. [Online]. Available: https://thingsboard.io/docs/reference/python-rest-client/. [Accessed 28 04 2022].

[34] INTERNET OF THINGER SL, "LINUX / RASPBERRY PI - Thinger.io Documentation," 2020. [Online]. Available: https://docs.thinger.io/linux. [Accessed 13 05 2022].

[35] The Thingsboard Authors, "Provision Device APIs," 2022. [Online]. Available: https://thingsboard.io/docs/pe/user-guide/device-provisioning/#provision-device-apis. [Accessed 13 05 2022].

[36] A. L. Bustamante, "Register a Device in the Console," 01 08 2015. [Online]. Available: https://community.thinger.io/t/register-a-device-in-the-console/23. [Accessed 13 05 2022].

[37] The Thingsboard Authors, "Server-side attributes," 2022. [Online]. Available: https://thingsboard.io/docs/pe/user-guide/attributes/#server-side-attributes. [Accessed 13 05 2022].

[38] INTERNET OF TIHNGER SL, "Add User Device," 2021. [Online]. Available: https://docs.thinger.io/server/api#add-user-device. [Accessed 13 05 2022].

[39] Open Group CAE Specification C309, DCE: Remote Procedure Call, Reading, United Kingdom: X/Open Company Ltd., U.K., 1994.

[40] INTERNET OF THINGER S.L., "Building the HTTP request," 2021. [Online]. Available: https://docs.thinger.io/http-devices#building-the-http-request. [Accessed 13 05 2022].

[41] The Thingsboard Authors, "REST API | ThingsBoard Community Edition," 2022. [Online]. Available: https://thingsboard.io/docs/reference/rest-api/. [Accessed 13 05 2022].

[42] The Thingsboard Authors, "HTTP Access Token based authentication | ThingsBoard Professional Edition," 2022. [Online]. Available: https://thingsboard.io/docs/pe/user-guide/ssl/http-access-token/. [Accessed 13 05 2022].

[43] The Thinsgboard Authors, "Telemetry upload API," 2022. [Online]. Available: https://thingsboard.io/docs/pe/reference/http-api/#publish-attribute-update-to-the-server. [Accessed 20 05 2022].

[44] INTERNET OF TINGER S.L., "Device Properties," 2021. [Online]. Available: https://docs.thinger.io/features/devices-administration#device-properties. [Accessed 20 05 2022].

[45] INTERNET OF THINGER S.L., "Device API," 2021. [Online]. Available: https://docs.thinger.io/features/devices-administration#device-api. [Accessed 20 05 2022].

[46] INTERNET OF TIHNGER S.L., "From device Write Call," 2021, [Online]. Available: https://docs.thinger.io/features/buckets#from-device-write-call. [Accessed 20 05 2022].

[47] The Thingsboard Authors, "Action Nodes | ThingsBoard Community Edition," 2022. [Online]. Available: https://thingsboard.io/docs/user-guide/rule-engine-2-0/action-nodes/. [Accessed 20 05 2022].

[48] INTERNET OF THINGER S.L., "USER ACCOUNTS - Thinger.io Documentation," 2021. [Online]. Available: https://docs.thinger.io/users-management. [Accessed 23 05 2022].

[49] INTERNET OF THINGER S.L., "PROJECTS MANAGER - Thinger.io Documentation," 2021. [Online]. Available: https://docs.thinger.io/projects. [Accessed 23 05 2022].

[50] The ThingsBoard Authors, "HTTP over SSL | ThingsBoard Community Edition," 2022. [Online]. Available: https://thingsboard.io/docs/user-guide/ssl/http-over-ssl/. [Accessed 23 05 2022].

[51] The Thingsboard Authors, "Advanced Role-Based Access Control (RBAC) for IoT devices and applications | ThingsBoard Professional Edition," 2022. [Online]. Available: https://thingsboard.io/docs/pe/user-guide/rbac/. [Accessed 23 05 2022].

[52] The Thingsboard Authors, "HTTP Device API Reference | ThingsBoard Professional Edition," 2022. [Online]. Available: https://thingsboard.io/docs/pe/reference/http-api/#publish-attribute-update-to-the-server. [Accessed 23 05 2022].

[53] INTERNET OF TIHNGER S.L., "OTA PROGRAMMING - Thinger.io Documentation," 11 2021. [Online]. Available: https://docs.thinger.io/extended-features/ota#firmware-upload-via-ota. [Accessed 23 05 2022].

[54] INTERNET OF THINGER S.L., "DEVICES - Thinger.io Documentation," 10 2021. [Online]. Available: https://docs.thinger.io/arduino. [Accessed 23 05 2022].

[55] The Thingsboard Authors, "Device Connectivity Protocols | ThingsBoard Community Edition," 2022. [Online]. Available: https://thingsboard.io/docs/reference/protocols/. [Accessed 23 05 2022].

[56] The Thingsboard Authors, "Queue processing pace," 2022. [Online]. Available: https://thingsboard.io/docs/user-guide/ota-updates/#queue-processing-pace. [Accessed 24 05 2022].

[57] The Thingsboard Authors, "White-labeling | ThingsBoard Professional Edition," 2022. [Online]. Available: https://thingsboard.io/docs/pe/user-guide/white-labeling/. [Accessed 24 05 2022].

[58] B. Nielsen, M. Banducci, J. A. Klode and P. Kalytta, "215465 – AX201 not using 2 receive streams," 28 04 2022. [Online]. Available: https://bugzilla.kernel.org/show_bug.cgi?id=215465. [Accessed 26 05 2022].

[59] T. Carpenter and 7signal, "MYSTERIES OF Wi-Fi ROAMING REVEALED - WHITEPAPER," 18 10 2017. [Online]. Available: https://cdn2.hubspot.net/hubfs/353374/Knowledge%20Base/MYSTERIES%20of%20Wi-Fi%20Roaming%20Revealed%20-%207SIGNAL%20Whitepaper.pdf. [Accessed 30 05 2022].

[60] Intel Corporation, "Wlan-Roaming-Aggressiveness-Einstellung," 28 10 2021. [Online]. Available: https://www.intel.de/content/www/de/de/support/articles/000005546/wireless/legacy-intel-wireless-products.html. [Accessed 30 05 2022].

[61] P. Kalytta, "215869 – iwlwifi: AX210: Device not roaming between APs," 22 04 2022. [Online]. Available: https://bugzilla.kernel.org/show_bug.cgi?id=215869. [Accessed 30 05 2022].

[62] IEEE Standards Association, LAN/MAN Standards Committee, "27.3.1.1 MU transmission," in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical*

Layer (PHY) Specifications - Amendment 1: Enhancements for High-Efficiency WLAN, New York, The Institute of Electrical and Electronics Engineers, Inc., 2021, p. 497.

[63] D. Coleman, "Trigger Frames," in *Wi-Fi 6 & 6E for dummies*, Hoboken, New Jersey, John Wiley & Sons, Inc., 2022, p. 24.

[64] Wifi Ninjas, "WN Blog 003 – WiFi 6 Deep Dive & Real World Testing," 03 07 2019. [Online]. Available: https://wifininjas.net/2019/07/03/wn-blog-003-wifi-6-deep-dive-real-world-testing/. [Accessed 01 06 2022].

[65] Wi-Fi Alliance, "Wi-Fi CERTIFIED™ Certificate - Certification ID: WFA101064," 08 10 2021. [Online]. Available: https://api.cert.wi-fi.org/api/certificate/download/public?variantId=104581. [Accessed 01 06 2022].

[66] Wi-Fi Alliance, "Wi-Fi CERTIFIED™ Certificate - Certification ID: WFA83471," 30 10 2020. [Online]. Available: https://api.cert.wi-fi.org/api/certificate/download/public?variantId=37184. [Accessed 01 06 2022].

[67] P. Lavoie and Wi-Fi Alliance, "Case 00163600," 2022.

[68] P. Kil, "Home · openremote/openremote Wiki · GitHub," 15 12 2021. [Online]. Available: https://github.com/openremote/openremote/wiki. [Accessed 13 05 2022].

[69] The Mainflux Contributors, "Overview - Mainflux," 07 03 2022. [Online]. Available: https://mainflux.readthedocs.io/en/latest/. [Accessed 13 05 2022].

[70] IEEE, "16.2.2 PPDU format," in *IEEE 802.11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, New York, IEEE, 2016, p. 2249f.

[71] IEEE, "Supplement To IEEE Standard For Information Technology-Telecommunications And Information Exchange - IEEE Std 802.11b-1999," 20 01 2000. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?punumber=6642. [Accessed 29 07 2019].

[72] J. M. Berg, "Radiotap - Defined Fields," 27 06 2019. [Online]. Available: http://www.radiotap.org/fields/defined. [Accessed 14 09 2019].

[73] P. Deutsch, "RFC 1952 - GZIP file format specification version 4.3," Aladdin Enterprises, 05 1996. [Online]. Available: https://tools.ietf.org/html/rfc1952. [Accessed 03 10 2019].

[74] konikofi, "Chasing Trigger Frames," 21 03 2021. [Online]. Available: https://konikofi.wordpress.com/2021/03/21/chasing-trigger-frames/. [Accessed 28 04 2022].

[75] P. Kalytta, "Einfluss von Beacon-Frames auf den Datendurchsatz in Wi-Fi-Netzen," Köln, 2019.

List of tables

List of figures

List of source code

Appendix

Appendix A: Comparison chart IoT-Management-Software

Maximum achievable points are defined by the requirement to reflect the importance: 10 points for MUST (required) requirements, 5 points for SHOULD (recommended) requirements, 3 points for MAY (optional) requirements and 1 point for nice-to-have (NITH) requirements. If a requirement is only partially achieved/supported, then only partial points are awarded. The online documentation of the individual solutions was used as a reference for this overview [68] [28] [24] [69]:

Category	**Maximum Points**	**Open Remote**	**Thinger.io**	**Things Board**	**Mainflux**
Provisioning					
SHOULD Initial setup can be automated via image	5	0	5	0	0
MUST Initial setup can be automated via script	10	10	10	10	5
SHOULD Pre-configuration possible directly from the software (setting and rolling out configuration parameters of the device)	5	5	5	5	2,5
MUST Preconfigure network connection for initial registration from remote site	10	0	10	5	0

Category	Maximum Points	Open Remote	Thinger.io	Things Board	Mainflux
MAY Display and capture of the roll-out status or general state of a device	3	3	3	3	0
MAY Deployment can be triggered via frontend	3	1,5	0	3	0
Authentication					
MUST Device Identity Management	10	10	10	10	10
MUST Secure authentication at registration	10	10	10	10	10
SHOULD Authorization of the devices during operation/actions	5	5	5	5	5
MAY Authentication via device identity possible	3	3	0	0	3
SHOULD Authentication parameters can be configured in the frontend	5	2,5	5	5	0
Configuration (Over-the-air programming)					

Category	Maximum Points	Open Remote	Thinger.io	Things Board	Mainflux
MUST Identification of devices possible (parameters such as location, network, device configuration)	10	10	10	10	10
SHOULD Automatic change of network connection after initial login (Automatic Device Configuration)	5	2,5	5	5	0
MUST Customization of functionality (network parameters such as changing the channel/radio parameters in operation).	10	5	10	10	0
MAY Bulk-Configuration	3	1,5	3	3	3
SHOULD Device grouping or configuration rules for rollouts	5	2,5	3,75	2,5	5
SHOULD Full configuration of the devices can be viewed and changed via frontend	5	2,5	5	5	0
Control					

Category	Maximum Points	Open Remote	Thinger.io	Things Board	Mainflux
SHOULD Remote control (SHELL or similar) or triggering of commands on the device.	5	2,5	5	5	0
MAY Change of the device state (Switched On/Switched Off/Connected/Disconnected)	3	1,5	3	3	0
SHOULD Trigger/automate reboots and updates (rolling upgrade)	5	2,5	5	5	0
MUST Start performance tests	10	5	10	10	0
Monitoring					
SHOULD Capture system metrics centrally (metadata such as location, OS and device version, update status, etc.)	5	5	5	5	5

Category	Maximum Points	Open Remote	Thinger.io	Things Board	Mainflux
MUST Centrally capture performance metrics (network health such as throughput, congestion, CTS/RTS status, packet loss, utilization (CPU etc.))	10	5	10	10	10
MUST Prepared presentation of metrics in the frontend (GUI)	10	5	10	10	0
SHOULD Reporting by the devices (Automated)	5	2,5	5	5	5
MAY Notification of errors or security breaches	3	0	0	3	3
SHOULD Automatic analyses/data visualization in dashboards	5	2,5	5	5	0
Security					
SHOULD mTLS or HTTPs, DTLS or similar possible for general communication	5	5	2,5	5	5

Category	Maximum Points	Open Remote	Thinger.io	Things Board	Mainflux
MUST Access tokens or certificate-based authentication	10	10	10	10	10
SHOULD RBAC or similar for users	5	2,5	0	5	0
NITH Multi-tenant capability	1	1	0	1	0
Diagnostics					
MUST Device condition detection	10	10	5	10	0
NITH Remote troubleshooting possible (Self-healing Network?)	1	0	0	0	0
MAY Audit-Logs	3	0	0	3	3
MAY Central logging	3	0	3	0	0
Up-to-dateness					
SHOULD System update	5	2,4	5	5	0
MAY Rollbacks	3	0	1,5	3	0
MUST Config backups	10	0	5	10	0
Total points	224	136,5	189,75	204,5	94,5

Appendix B: Data References

Description	Link
Measurement Data (InfluxDB):	https://www.kalytta.net/th-assets/master/.wifi-influx-data.tar.gz
Wireshark packet captures created while testing for OFDMA functionality, containing OFDMA Trigger Frames	https://www.kalytta.net/th-assets/master/pcaps/
Further graphs with measurements for throughput in 6 GHz and 5 GHz 802.11ax and 802.11ac	https://www.kalytta.net/th-assets/master/graphs/

Appendix C: Figures

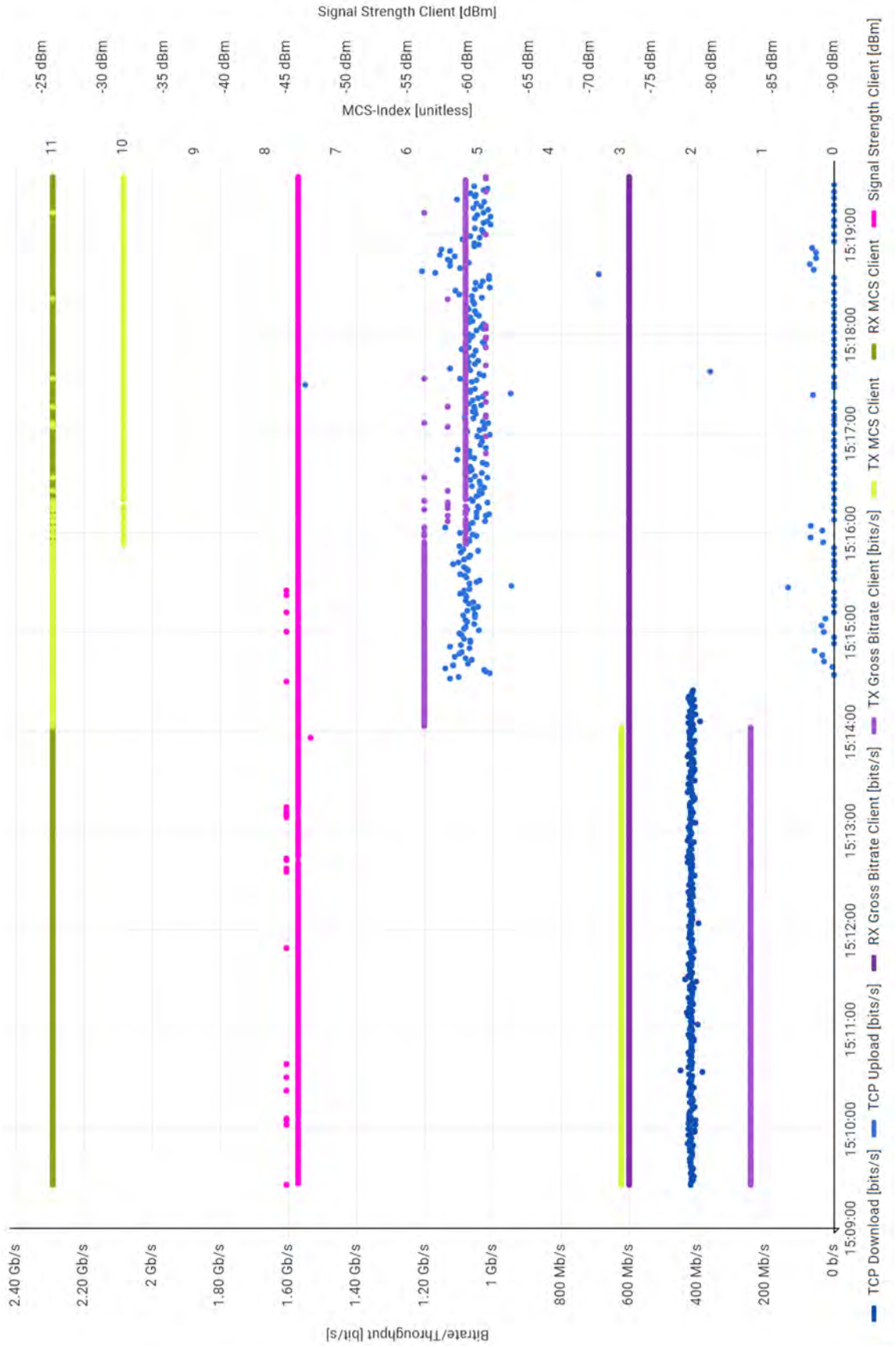

Figure 25: Measurement over 6 GHz with 80 MHz channel width, TCP: Some data points in the upload are zero.

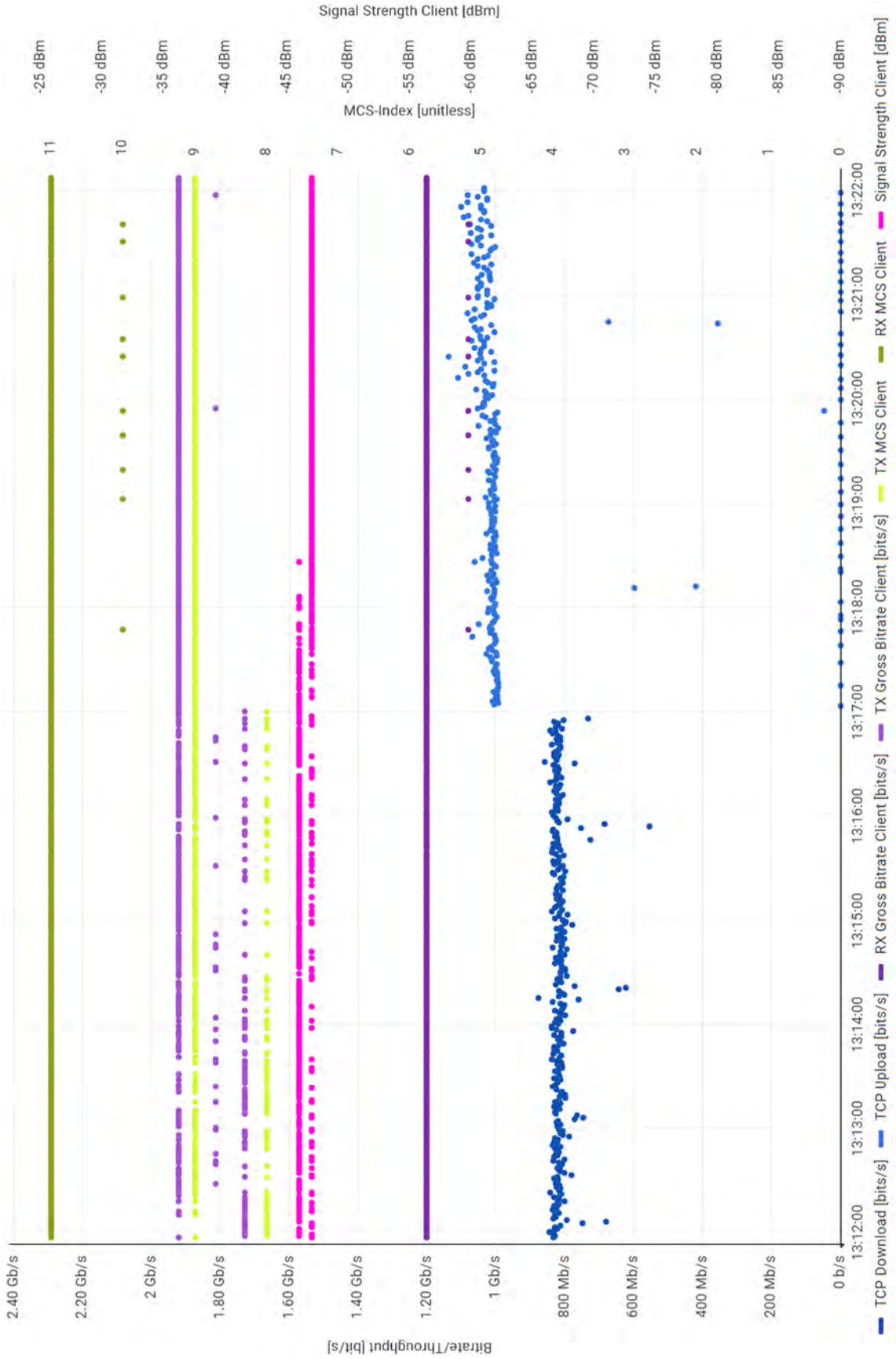

Figure 26: Measurement over 6 GHz with 160 MHz channel width, TCP. Download with one spatial stream, upload with two spatial streams. Throughput reaches over 1 Gbit/s in this case.

Figure 27: Corresponding RTT measurement. The sawtooth-like behavior of while downloading with high jitter can be clearly differentiated from the low jitter behavior while uploading from the client.

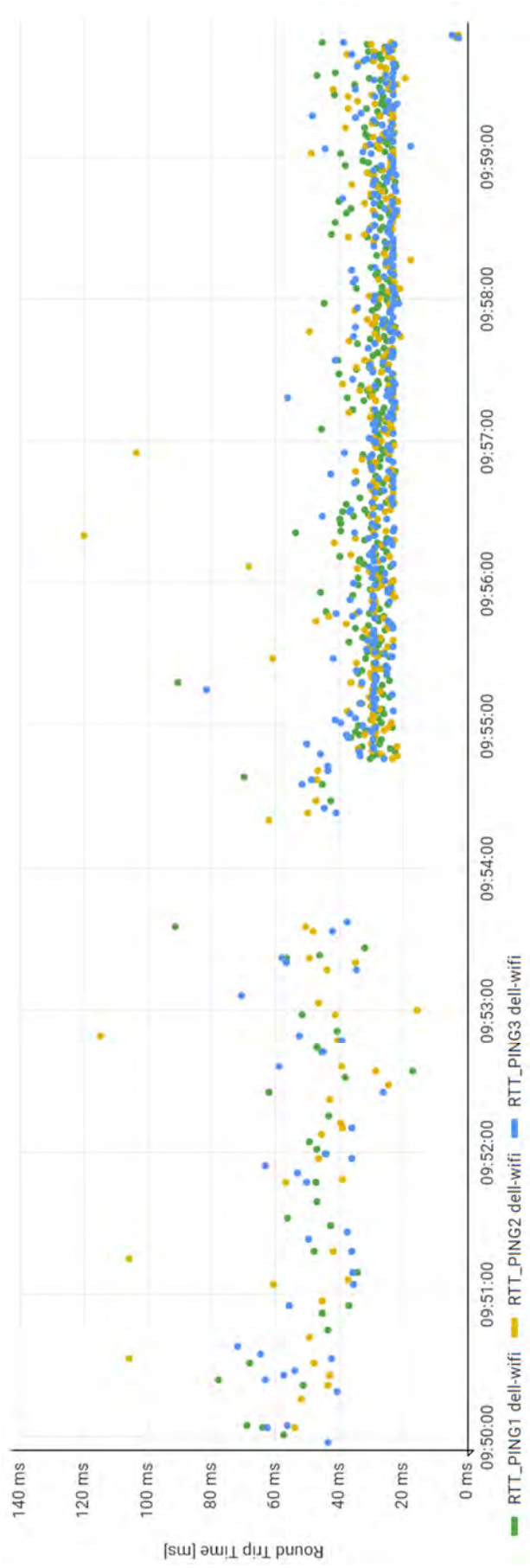

Figure 28: RTTs measured via ICMP Ping for a UDP measurement, the first half for download, the second half for upload from client.

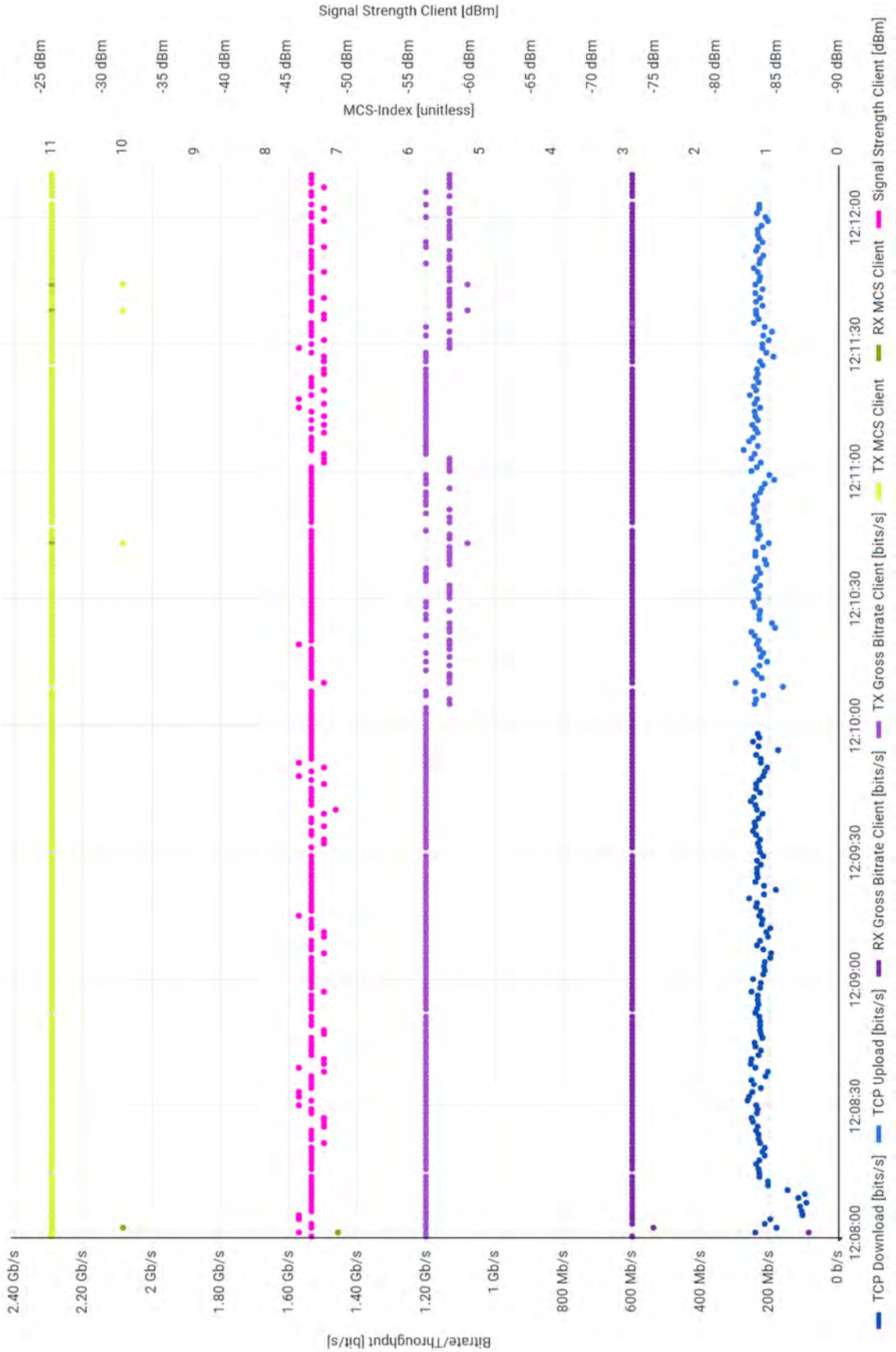

Figure 29: Measurement over 6 GHz with 80 MHz channel width, TCP. With two spatial streams in the TX theoretical throughput is 1.2 Gbit/s, due to limits by the RX spatial stream (only one), real throughput is much lower (only about 230 Mbit/s)

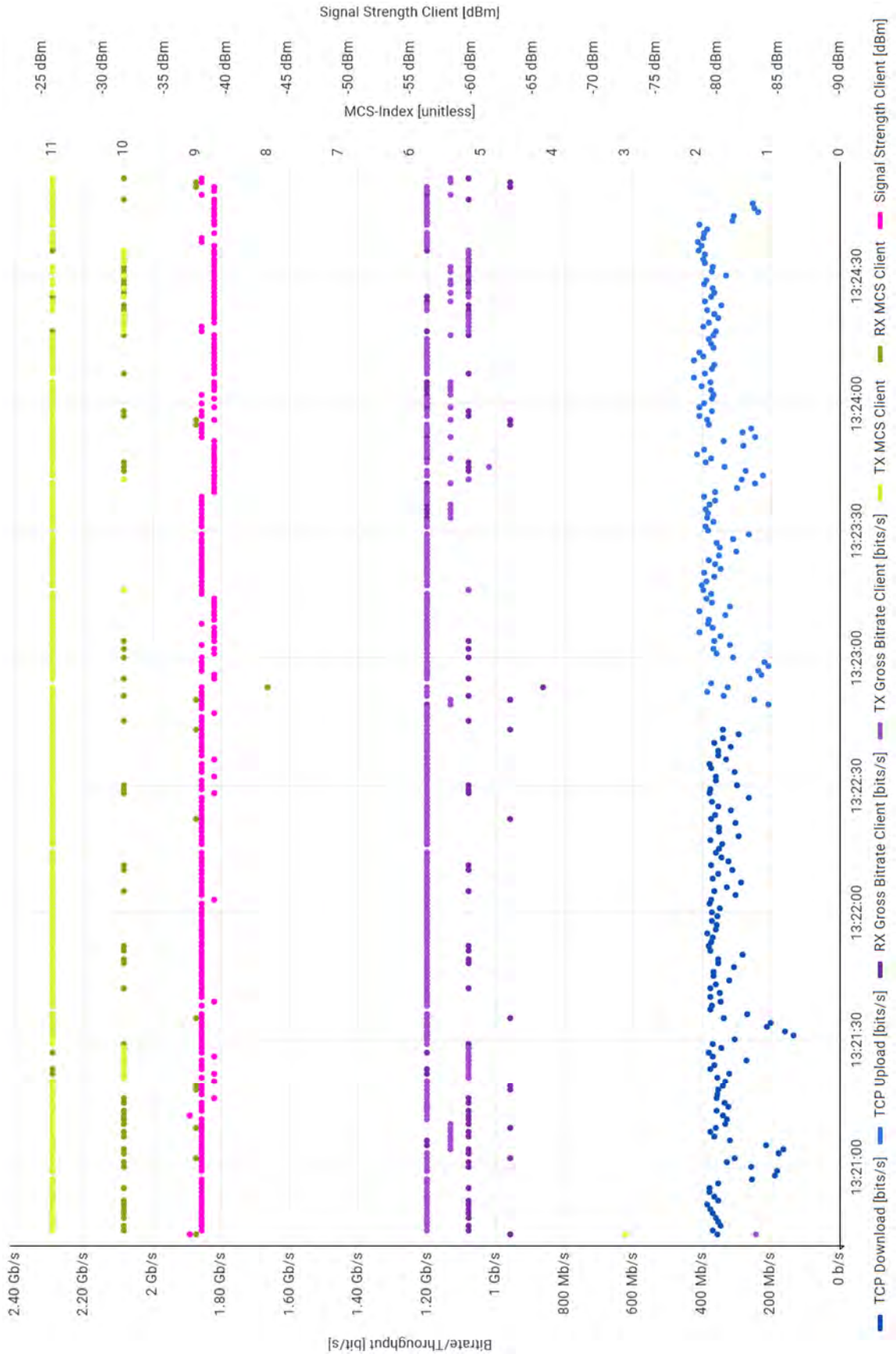

Figure 30: Measurement over 5 GHz with 80 MHz channel width, TCP. Throughput is nearly doubled with about 400 Mbit/s. The MCS flapped between 10 and 11 corresponding to 1080 Mbit/s and 1201 Mbit/s for both RX and TX.

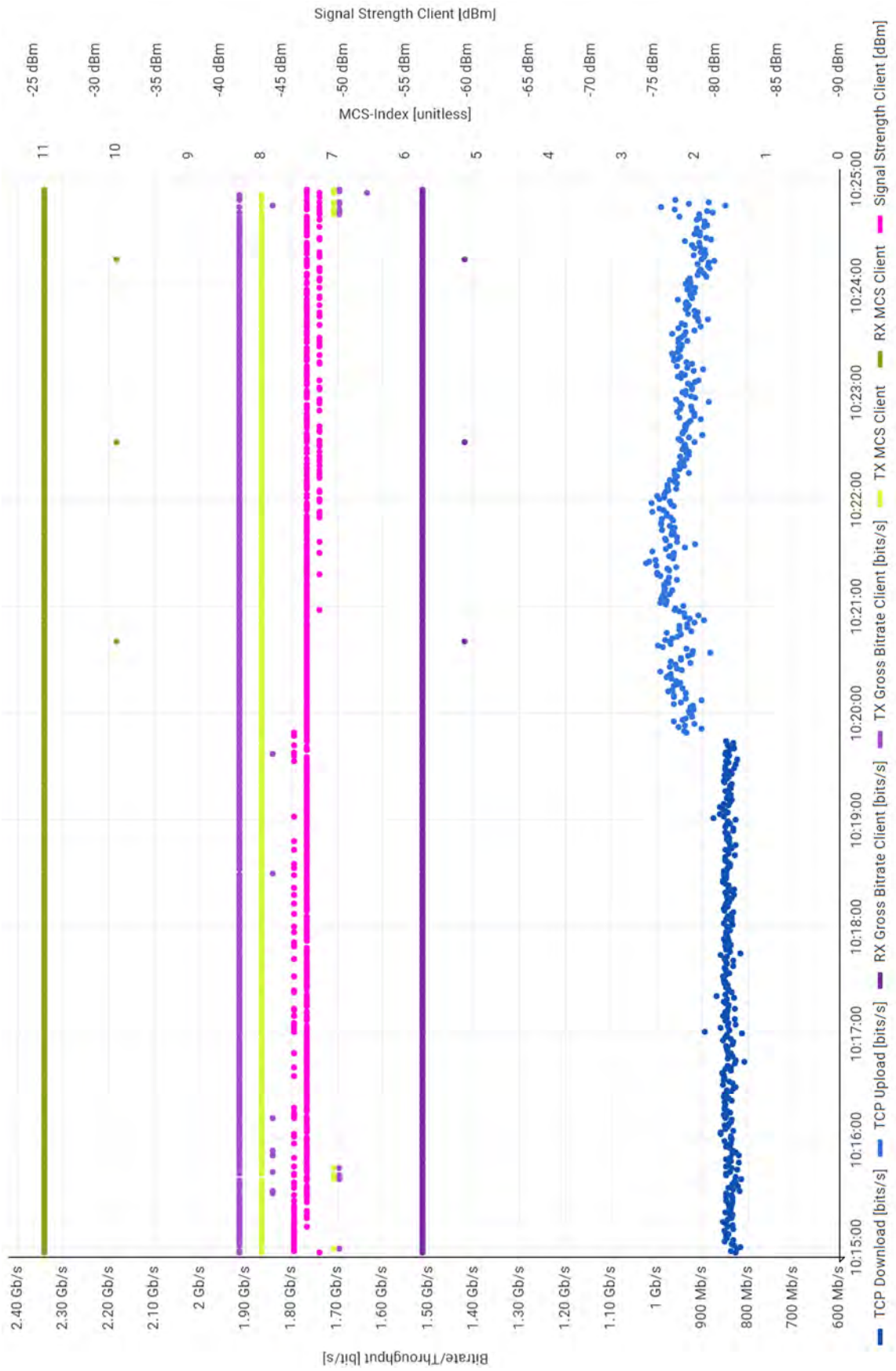

Figure 31: Measurement over 6 GHz with 160 MHz channel width, TCP. Transmit via two spatial streams reaches about 1 Gbit/s. Gross bitrates are wrongly reported here by netlink, RX MCS 11 corresponds to 1201 Mbit/s instead of 1500 Mbit/s and TX MCS 8 corresponds to 1729 Mbit/s

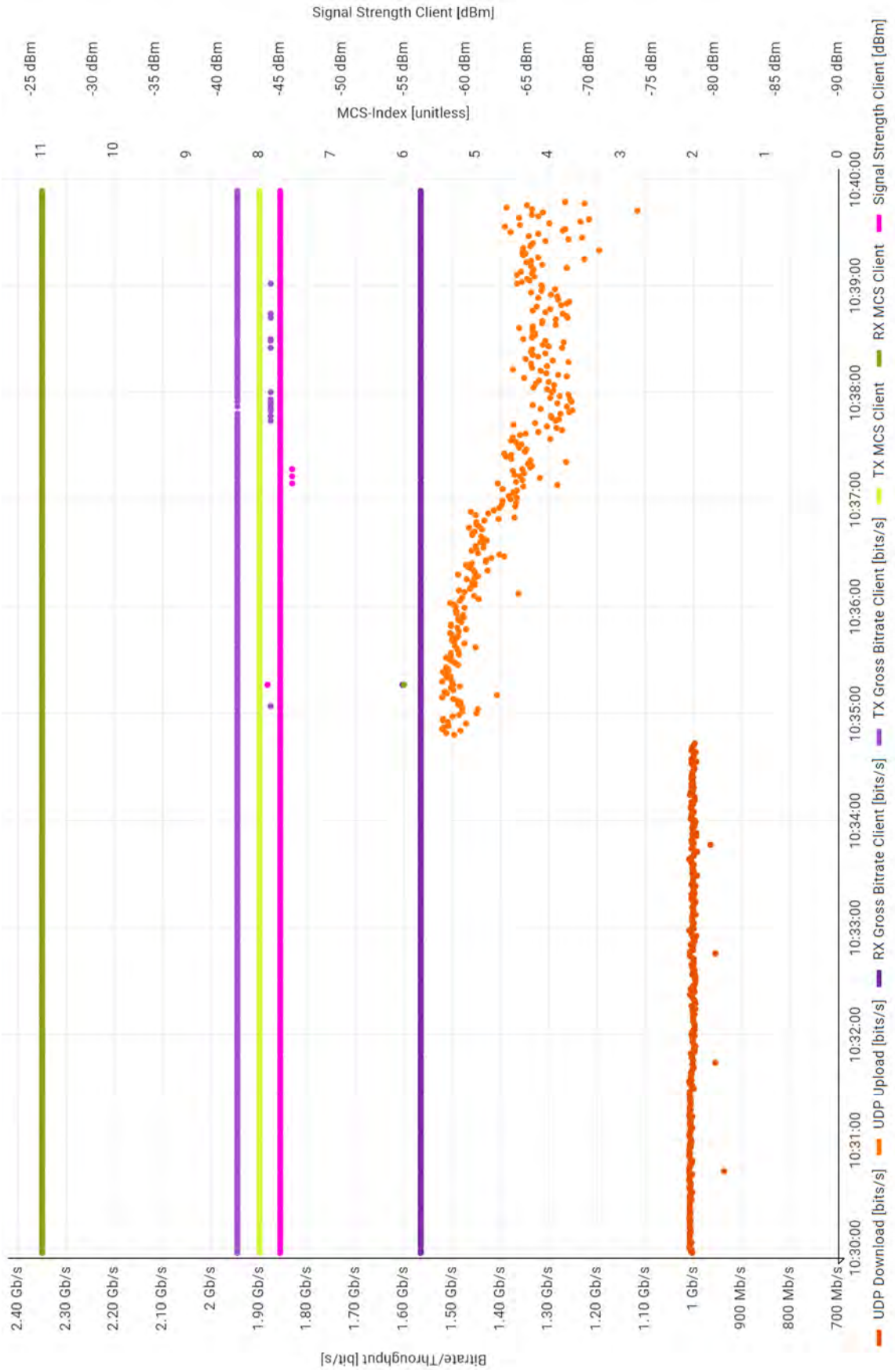

Figure 32: Measurement over 6 GHz with 160 MHz channel width, UDP. Transmit throughput reaches over 1.5 Gbit/s. Gross bitrates are wrongly reported here by netlink, RX MCS 11 corresponds to 1201 Mbit/s instead of 1580 Mbit/s and TX MCS 8 corresponds to 1729 Mbit/s

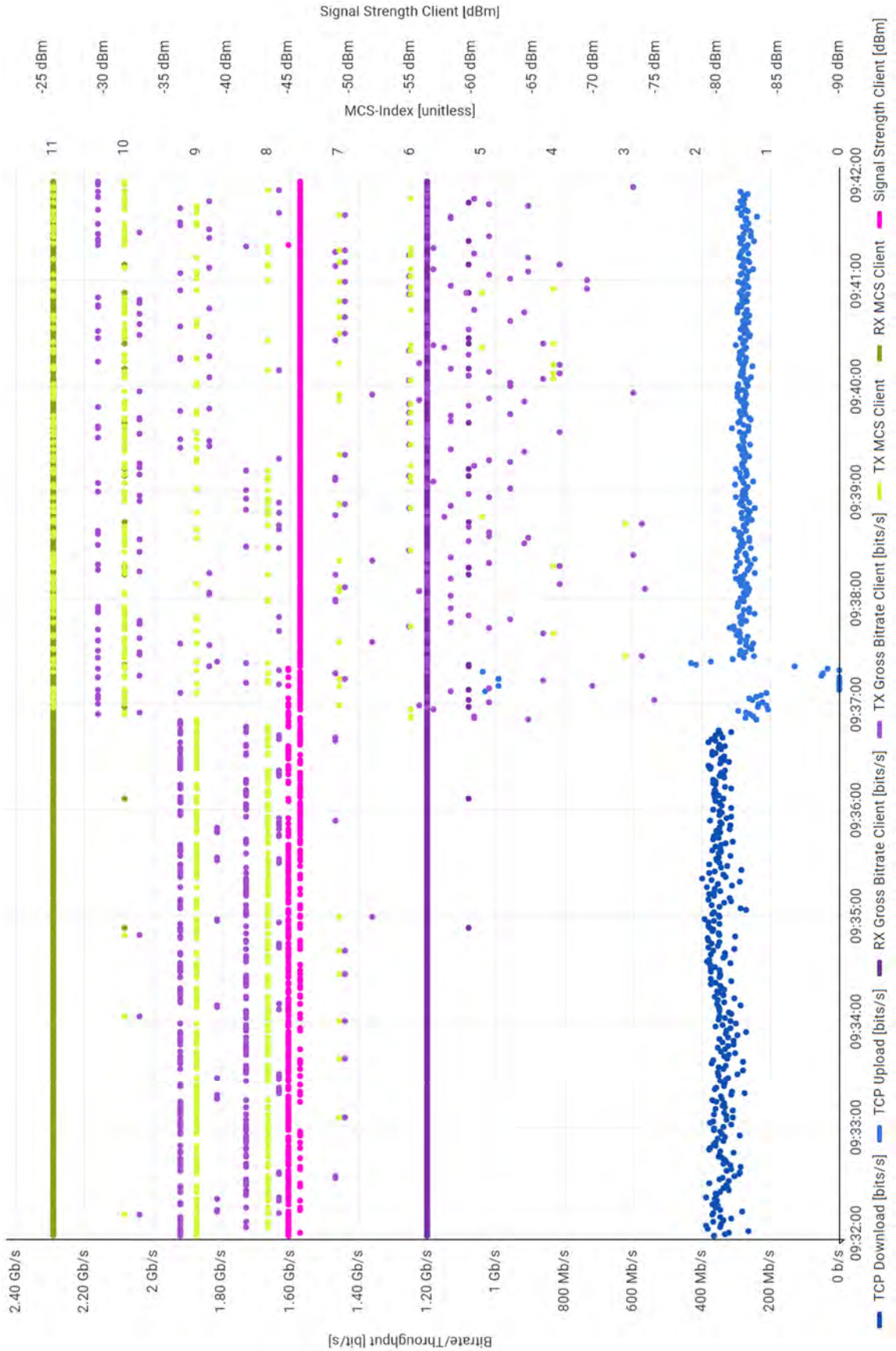

Figure 33: Measurement over 6 GHz with 160 MHz channel width, TCP, between two Clients in two different BSS of an ESS. There are obvious fluctuations in the device transmit bitrates. RX and TX throughput are very low with about 300-400 Mbit/s compared to the expected 1 Gbit/s.

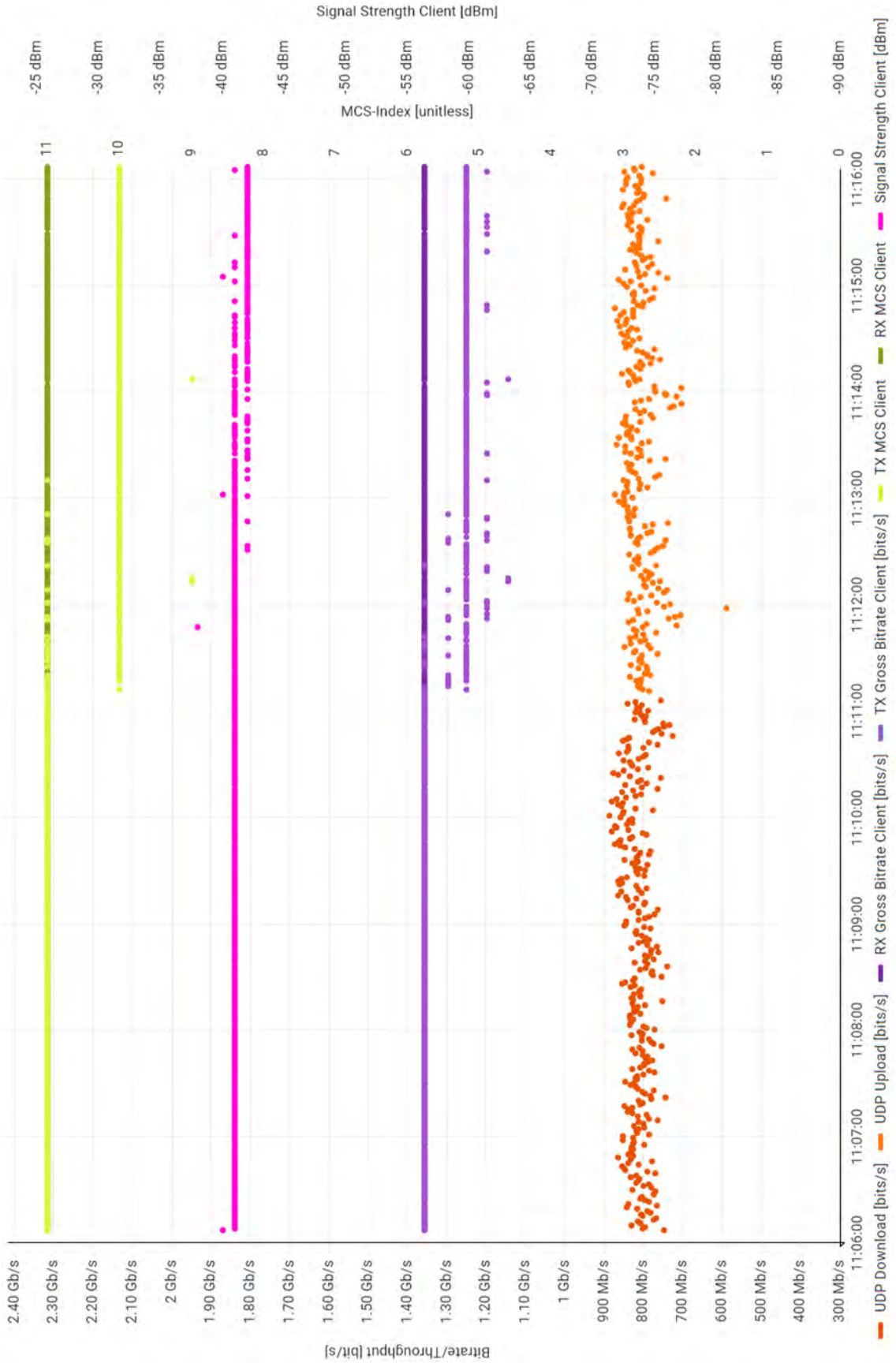

Figure 34: Measurement over 5 GHz with 80 MHz channel width, UDP, between two Clients in two different BSS of an ESS. Throughput reaches 850 Mbit/s which is reduced compared to the 900-1000 Mbit/s reached with an external measurement server.

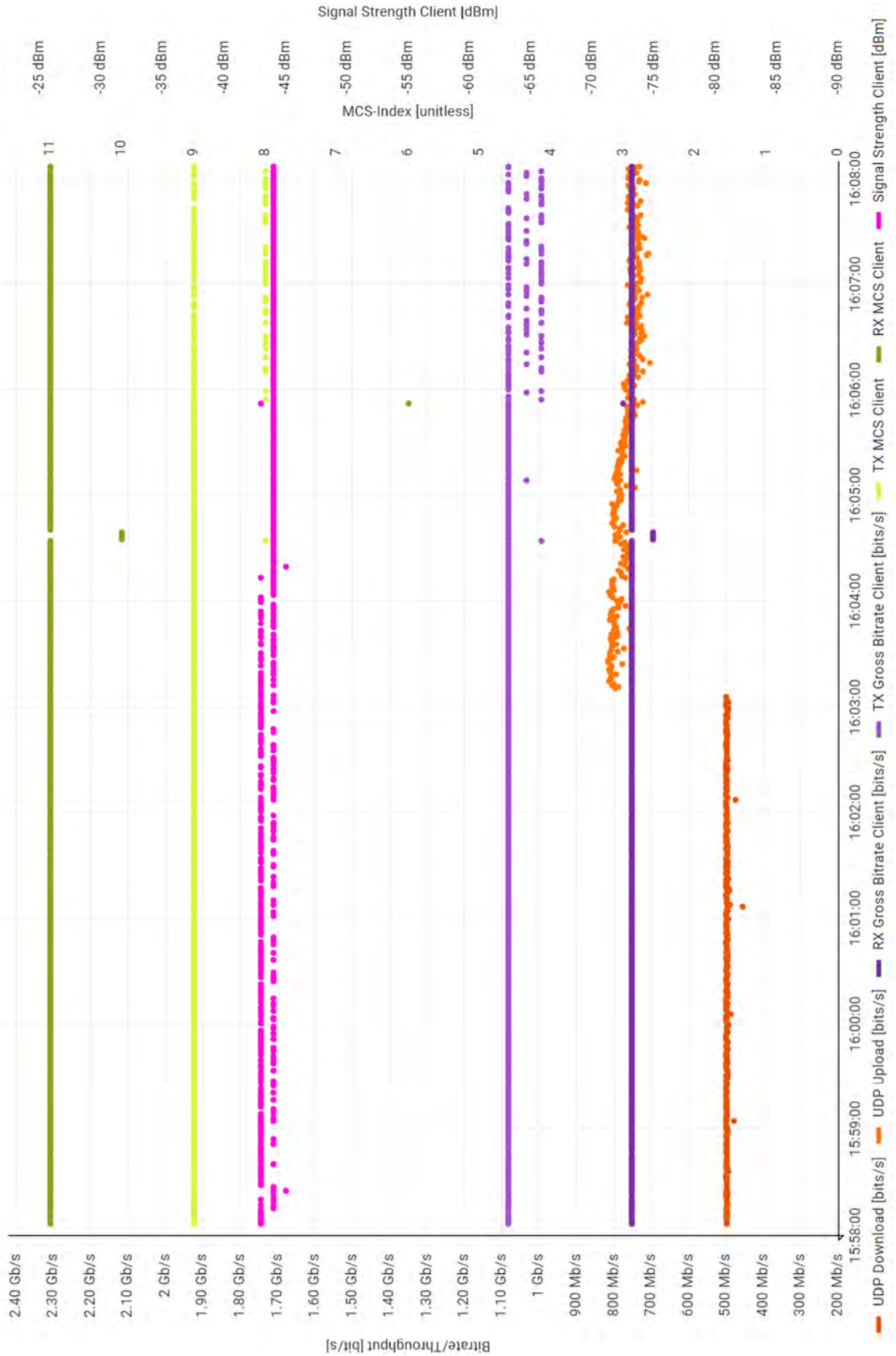

Figure 60: Measurement over 6 GHz with 80 MHz channel width, UDP. Due to only one spatial stream in the download direction, download throughput is not directly comparable to download throughput with 5 GHz, which uses two spatial streams

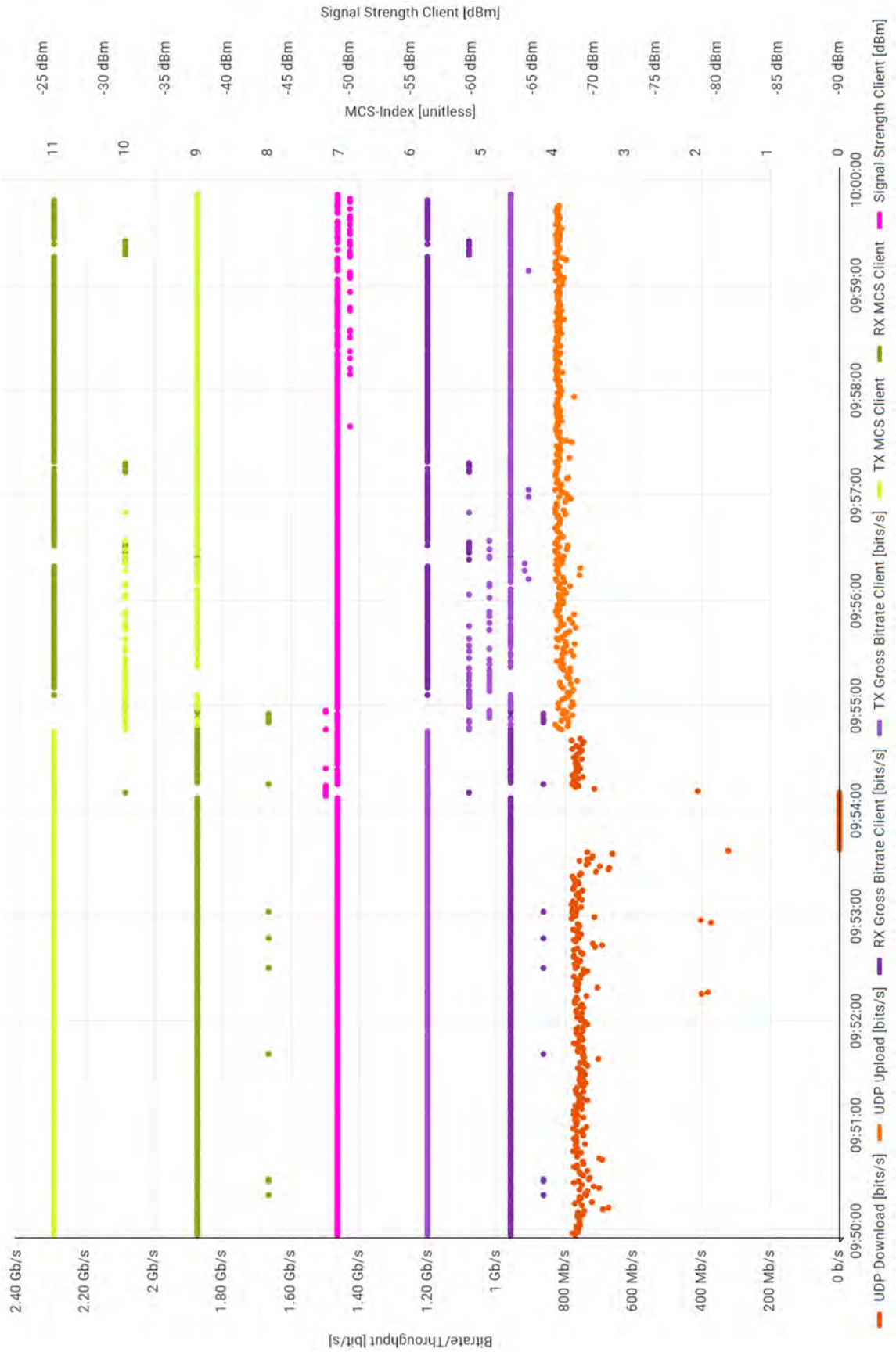

Figure 61: Measurement over 5 GHz with 80 MHz channel width and 802.11ax enabled, UDP. Throughput is like 6 GHz upload in both directions, due to using two spatial streams. Note the change in the selected MCS/bitrate when actually transmitting or receiving and it changing when the direction is not in use.

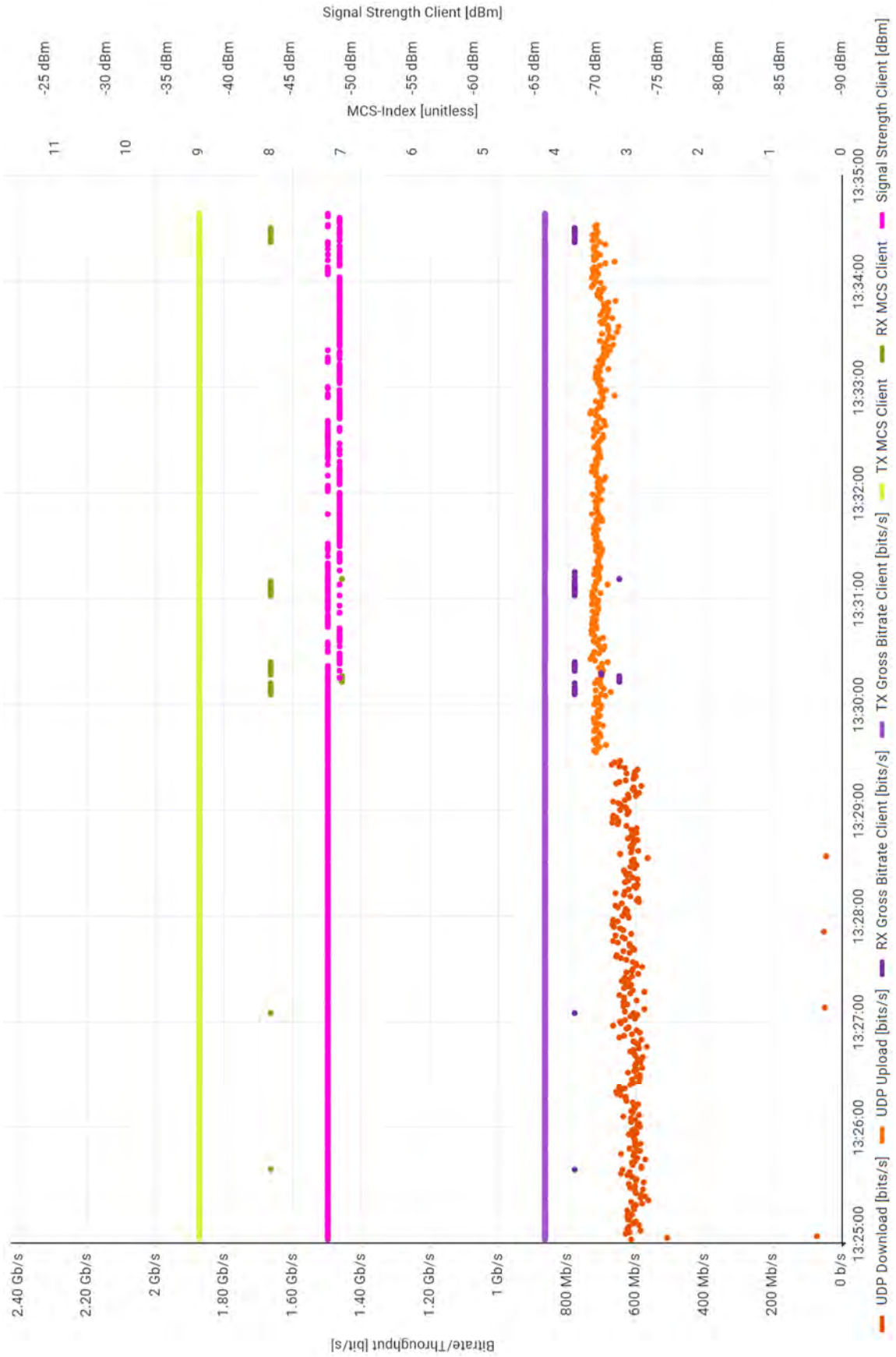

Figure 62: Measurement over 5 GHz with 80 MHz channel width and 802.11ac. Bitrate is lower despite same MCS as in the figure above due to different OFDM characteristics.

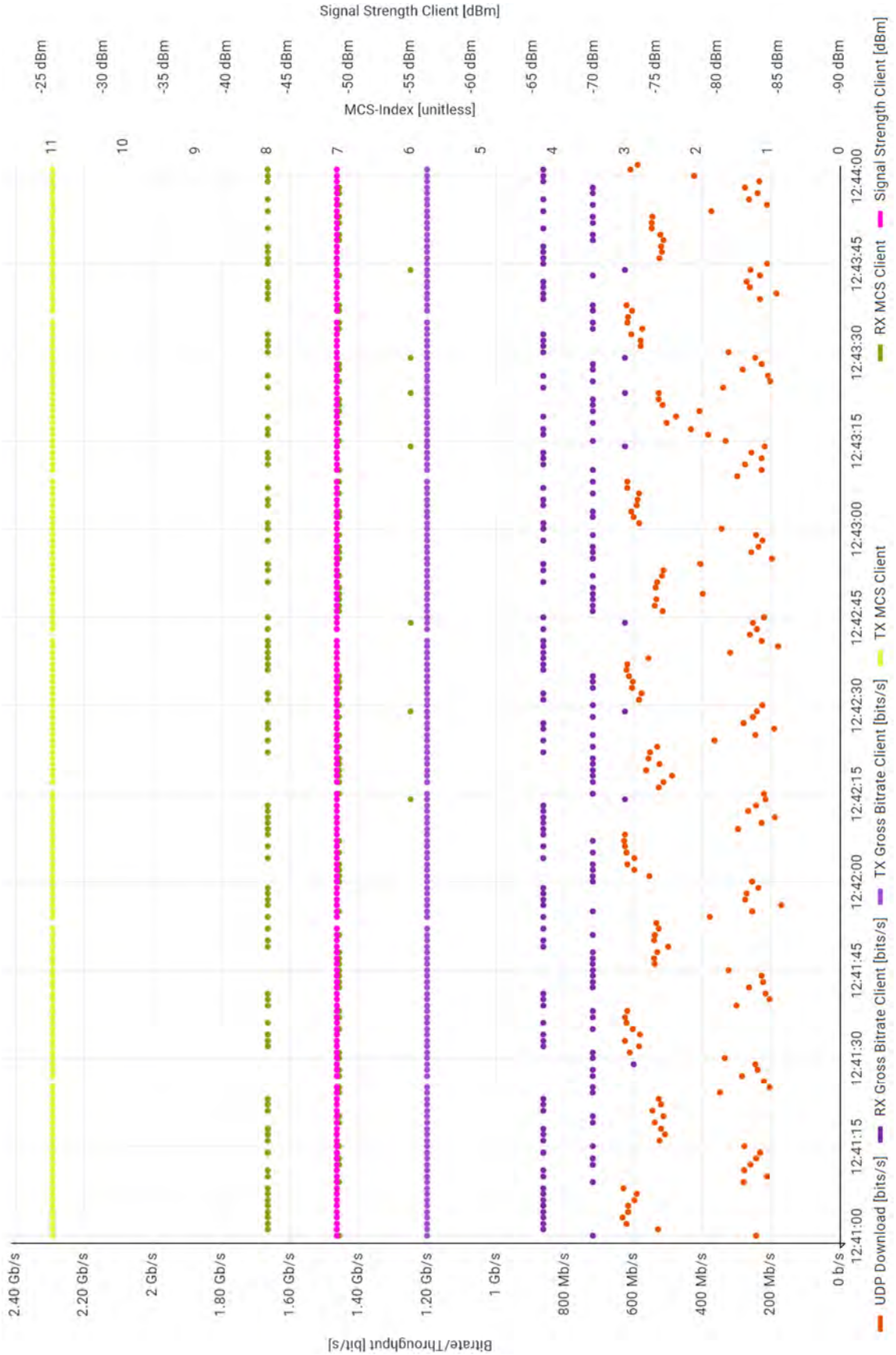

Figure 63: NetworkManager causes low throughput while scanning on the interface for seven second intervals.

Eidesstattliche Erklärung

Ich versichere hiermit, die vorgelegte Arbeit in dem gemeldeten Zeitraum ohne fremde Hilfe verfasst und mich keiner anderen als der angegebenen Hilfsmittel und Quellen bedient zu haben.

Köln, den 16. Juni 2022

Unterschrift

Philipp Kalytta