VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

VYUŽITÍ CLOUDOVÝCH SLUŽEB PRO PŘÍSTUP K VESTAVĚNÝM IOT ZAŘÍZENÍM

DEPLOYMENT OF CLOUD SERVICES FOR EMBEDDED IOT DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VÍT SOLČÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VÁCLAV ŠIMEK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Solčík Vít**
Program: Informační technologie
Název: **Využití cloudových služeb pro přístup k věstavěným IoT zařízením**
Deployment of Cloud Services for Embedded IoT Devices
Kategorie: Vestavěné systémy

Zadání:

1. Zabývejte se problematikou cloudových služeb a možností jejich využití v kombinaci s vestavěnými IoT zařízeními.
2. Připravte rešerši aktuálních trendů v této oblasti a cloudových služeb. Do rešerše zahrňte minimálně AWS IoT, IBM Natson IoT, AZURE IoT a Google Cloud IoT. Na základě zjištěných skutečností se blíže zaměřte na jednu z dostupných cloudových služeb.
3. Navrhněte koncept demonstrační aplikace, v níž by bylo využito vlastností zvolené cloudové služby. Pro daný scénář zvolte vhodný typ platformy vestavěného IoT zařízení.
4. Implementujte obslužný firmware, který zvolenému IoT zařízení umožní komunikovat s cloudovou službou prostřednictvím rozhraní Ethernet.
5. Vytvořte demonstrační aplikaci dle navrženého konceptu z bodu 3. zadání, díky níž bude možné prostřednictvím cloudové služby názorným způsobem vizualizovat data poskytnutá zvolenou IoT platformou.
6. Zhodnoťte dosažené výsledky a navrhněte možností dalšího rozšíření či vylepšení.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Šimek Václav, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 31. října 2018

Abstrakt

Tato práce si klade za cíl provést analýzu stávajících přístupů k realizaci cloudových služeb pro oblast IoT a navrhnout vlastní řešení demonstrující tohoto konceptu v prostředí vestavěných zařízení s omezenými zdroji. Pozornost je taktéž věnována aspektům zabezpečení TCP/IP komunikace mezi cloudovou službou a vestavěným zařízením za použití rozhraní Ethernet. Nedílnou součástí navrhovaného řešení je taktéž implementace potřebných částí firmware pro komunikaci s cloudovou platformou IoT pomocí podporovaných síťových protokolů. V neposlední řadě se tato práce zabývá i návrhem a praktickou realizací demonstrační aplikace, která využije výsledné komunikační infrastruktury.

Abstract

Main purpose of this thesis is to prepare an analysis of the existing approaches to the realization of cloud-based services for IoT domain and propose a solution demonstrating the exploitation of this concept for embedded devices with constrained resources. Departing from this point, the attention is further given to the aspect of secure TCP/IP communication between cloud service and embedded device using Ethernet interface. An integral part of the proposed solution is also comprising the implementation of necessary firmware for interact with IoT platform via supported network protocols. Last but not least this work is dealing as well with the design and practical deployment of a demonstration application, which uses this communication infrastructure.

Klíčová slova

IoT, cloud, internet věcí, vestavěné zařízení, Azure, Amazon Web Services, AWS, IBM Watson, Google Cloud IoT

Keywords

IoT, cloud, internet of things, embedded device, Azure, Amazon Web Services, AWS, IBM Watson, Google Cloud IoT

Citace

SOLČÍK, Vít. *Využití cloudových služeb pro přístup k vestavěným IoT zařízením.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

Využití cloudových služeb pro přístup k vestavěným IoT zařízením

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka. Další informace mi poskytla firma NXP Semiconductors Czech Republic, spol. s r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vít Solčík
15. května 2019

Poděkování

Poděkování náleží panu Ing. Václavu Šimkovi za vedení práce a firmě NXP Semiconductors Czech Republic, spol. s r.o. za poskytnutí zázemí a prostředků potřebných k vývoji.

Obsah

Kapitola 1

Úvod

Internet of things (chcete-li méně používaným českým překladem – Internet věcí) zažívá poslední roky nepopiratelný rozmach. Zájem o tuto technologii s sebou přináší nové výzvy a rizika, kterým se věnuje tato práce.

Nároky na jednoduchost a snadné použití je zcela protichůdný neméně důležitému faktoru bezpečnosti. V důsledku rostoucí obliby a rozšíření IoT (Internet of Things) napříč různými odvětvími, jako je zdravotnictví a doprava, je této technologii přiřazena důležitá úloha, kdy je kladen důraz na spolehlivost. Zkombinování těchto požadavků a dodání uspokojivé technologie nyní zaměstnává největší technologické firmy jako je Microsoft, IBM, Google a Amazon, které již přišly s několika nástroji a mechanismy, které by měly požadavky splňovat.

Tato práce se zabývá zprostředkováním a využitím těchto služeb na vestavěných zařízeních, které se díky své nízké pořizovací ceně, značné modularitě a dostatečnému výkonu jeví jako cílová platforma pro použití IoT.

Vestavěné zařízení mohou, ale ve většině případů nedisponují operačním systémem (alespoň ne v takové podobě, jak ho zná většina uživatelů). Tato skutečnost implikuje, že jejich obsluha a programové vybavení je více spjato s konkrétním HW (hardware), na kterém je zamýšleno provozovat danou technologii. Z tohoto důvodu je nutno koncovému uživateli nabídnout již existující řešení, které může převzít a snadno upravit s ohledem na svoje konkrétní potřeby či očekávaný způsob nasazení.

V kapitole 2 jsou probrány existující platformy podporující IoT komunikaci, které jsou na konci této kapitoly porovnány z pohledu využitelnosti v prostředí vestavěných systémů. Kapitola 3 pojednává o síťové komunikaci vestavěného zařízení prostřednictvím rozhraní Ethernet, na kterou navazuje kapitola 4 mapující využití kryptografie, za účelem šifrované komunikace. Konkrétní cloudová platforma využita pro účely IoT komunikace se zvoleným typem vestavěného zařízení (vývojová deska FRDM-K64F) je popsána v kapitole 5, která je také využita v kapitole 6 pro účely demonstrační aplikace a jejího testování. Kapitola 8 obsahuje rozbor zabezpečení a hrozeb, které je v tomto kontextu nutno zvážit spolu s nástinem možného směrovaní vývoje. Dosažené výsledky jsou shrnuty v kapitole 9.

Kapitola 2

Současné cloudové IoT platformy

Tato kapitola mapuje v současné době existující platformy různých poskytovatelů, které zprostředkovávají IoT komunikaci a s ní spojené služby.

Každý z těchto poskytovatelů, přestože nabízí z pohledu uživatele stejnou službu, přistupuje k problematice odlišně. Nicméně tyto rozdíly se projevují až při zpracování samotných zpráv a dat na straně serveru. Z pohledu komunikujícího klienta (v našem případě vestavěného zařízení) se zmiňované služby až na některé implementační detaily do značné míry neliší. Tudíž se postřehy a dosažené výsledky dají s poměrně malým úsilím přenést napříč všemi platformami zmíněnými v této práci.

Z této nabyté skutečnosti je v implementační části této práce použito pouze jednoho z nabízených řešení komerční sféry – a to Microsoft Azure.

2.1 Amazon Web Services (AWS)

Amazon Web Services poskytuje oproti konkurenci patrně nejširší podporu pro vývoj IoT na vestavěném zařízení díky obsáhlé dokumentaci a připravenému SDK (software development kit) v jehož rámci je připraveno pár ukázkových aplikací, jež je možné využít při tvorbě vlastního řešení. Tyto aplikace jsou psány v jazyce C a předpokládají běh na systému Linux, což není u vestavěných zařízení typické. Nicméně dokumentace obsahuje návod popisující kroky nutné pro přepsání API (Application Programming Interface – rozhraní pro programování aplikace), kterými docílíme přenesení aplikace na platformu jinou. Tyto kroky sestávají hlavně z nahrazení funkcí obsluhující tyto služby:

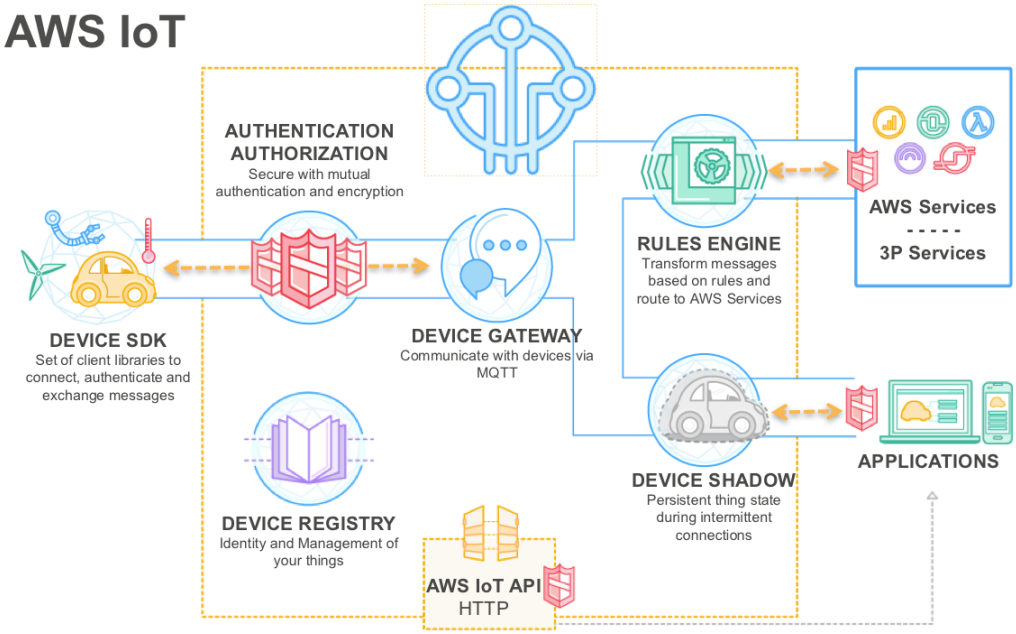
- Funkce časovače
- Síťové funkce
- Funkce jednoho či více vláknového řízení (threading)

Nastavení serverové strany vyžaduje registraci účtu AWS (Amazon Web Services), kde je nutné vytvořit identitu zařízení nazvané v rámci AWS "thing". Pro zařízení je vytvořeno bezpečnostní oprávnění, vygenerován certifikát a klíč, které je následně nutné přenést na vestavěné zařízení. Celý tento proces lze bez větších problému replikovat za užití návodů, jež jsou k dispozici na webových stránkách Amazonu [3].

Po absolvování prerekvizit popsaných výše je již serverová strana připravena přijímat autentizovanou komunikaci z daného zařízení. Hierarchie zpráv je členěna do topics (i když sám výrobce neposkytuje překlad, nejvhodnější alternativa je patrně 'téma'), ke kterým se

jednotlivá zařízení musí pro příjem i odesílání zpráv přihlásit (subscribe) a pomocí pravidel (rules) jsou zprávy zpracovány a arbitrovány na příslušný koncový bod. Tuto hierarchii můžeme vidět na obr. 2.1.

Z obrázku níže je také patrné, že pro komunikaci zařízení s cloudovým serverem je k dispozici pouze síťový protokol MQTT.



Obrázek 2.1: **Architektura komunikace AWS.** Zprávy jednotlivých zařízení jsou po příchodu na bránu (gateway) rozesílány dál na základě pravidel (rules) [4].

2.2 IBM Watson

IBM Watson IoT se principiálně zásadně neliší od platformy AWS probrané v předchozí kapitole. Nicméně podpora pro vývoj na vestavěných zařízeních není tolik obsáhlá. Nastavení serverové strany předpokládá registraci uživatelského účtu, vytvoření Watson IoT Platform instance (pro účely vývoje a „osahání“ je k dispozici bezplatná varianta), registrace samotného zařízení a vytvoření autentizačního tokenu pro účely zabezpečené komunikace.

IBM na svých stránkách odkazuje na poněkud zastaralý a dlouho neaktualizovaný repozitář [17], ve kterém se nachází demonstrativní aplikace psaná též v jazyce C. Nicméně součástí aplikace je externí knihovna pro komunikaci prostřednictvím MQTT využívající systémových knihoven, které nejsou na vestavěných zařízeních vetšinou podporovány. Použití tohoto ukázkového kódu by vyžadovalo vlastní analýzy využitelnosti na vestavěném zařízení a patrně širší zásah do celé aplikace.

Výhodou zůstává, že cloudové řešení od IBM, je schopné komunikovat (pro IoT standardním protokolem) pomocí MQTT, ale také protokolem HTTP – respektive HTTPS při šifrované komunikaci pomocí TLS (Transport Layer Security).

2.3 Google Cloud IoT

Google Cloud IoT je poměrně nová platforma, což podtrhuje i fakt, že v době začátku implementační části práce tato platforma ještě neexistovala (alespoň ne v podobě jakou zaujímá dnes). Koncepčně je opět snadné najít korelaci mezi touto a předchozími platformami. Stejně jako u AWS je komunikace tříděna pomocí topiců (témat) a přihlášení (subscribe) k nim. Nastavení serverové strany probíhá skrze Cloud Shell příkazovou řádku přístupnou přes webové rozhraní.

Celý proces zahrnuje vytvoření topic 2.2, vytvoření subscription 2.3, vytvoření registru zařízení 2.4 a vygenerování RSA klíčů 2.5. Pro testovací účely byla vytvořena instance simulačního zařízení 2.6 následována odesláním 2.7 a vyzvednutím 2.8 testovacích zpráv.

- Vytvoření topic

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to unified-surf-238118.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
vit_solcik@cloudshell:~ (unified-surf-238118)$ gcloud pubsub topics create my-topic
Created topic [projects/unified-surf-238118/topics/my-topic].
vit_solcik@cloudshell:~ (unified-surf-238118)$
```

Obrázek 2.2: **Vytvoření topic v Google Cloud Shell**

- Vytvoření subscription k topic

```
vit_solcik@cloudshell:~ (unified-surf-238118)$ gcloud pubsub subscriptions create projects/unified-surf-238118/s
ubscriptions/my-subscription --topic=my-topic
Created subscription [projects/unified-surf-238118/subscriptions/my-subscription].
vit_solcik@cloudshell:~ (unified-surf-238118)$
```

Obrázek 2.3: **Vytvoření subscription k topic v Google Cloud Shell**

- Vytvoření registru zařízení (device registry)

```
vit_solcik@cloudshell:~/nodejs-docs-samples/iot (unified-surf-238118)$ gcloud iot registries create my-registry
--project=unified-surf-238118 --region=us-central1 --event-notification-config=topic=projects/unified-surf-23811
8/topics/my-topic
Created registry [my-registry].
```

Obrázek 2.4: **Vytvoření registru zařízení v Google Cloud Shell**

- Vygenerování RSA klíčů pro zařízení

```
vit_solcik@cloudshell:~/nodejs-docs-samples/iot (unified-surf-238118)$ ./scripts/generate_keys.sh
Generating a RSA private key
..............................+++++
.................................................+++++
writing new private key to 'rsa_private.pem'
-----
read EC key
writing EC key
```

Obrázek 2.5: **Vygenerování RSA klíčů pro zařízení v Google Cloud Shell**

- Vytvoření simulačního zařízení

```
vit_solcik@cloudshell:~/nodejs-docs-samples/iot (unified-surf-238118)$ gcloud iot devices create my-node-device
--project=unified-surf-238118 --region=us-central1 --registry=my-registry --public-key path=rsa_cert.pem,type=rs
256
Created device [my-node-device].
```

Obrázek 2.6: **Vytvoření simulačního zařízení v Google Cloud Shell**

- Odeslání testovacích zpráv ze simulovaného zařízení

```
vit_solcik@cloudshell:~/nodejs-docs-samples/iot/mqtt_example (unified-surf-238118)$ node cloudiot_mqtt_example_n
odejs.js mqttDeviceDemo --cloudRegion=us-central1 --projectId=unified-surf-238118 --registryId=my-registry --dev
iceId=my-node-device --privateKeyFile=../rsa_private.pem --numMessages=25 --algorithm=RS256 --mqttBridgePort=443
Google Cloud IoT Core MQTT example.
connect
Publishing message: my-registry/my-node-device-payload-1
Config message received:
Config message received:
Publishing message: my-registry/my-node-device-payload-2
Publishing message: my-registry/my-node-device-payload-3
```

Obrázek 2.7: **Odeslání testovacích zpráv ze simulovaného zařízení v Google Cloud Shell**

- Vyzvednutí zpráv daného subscribtion

```
vit_solcik@cloudshell:~/nodejs-docs-samples/iot/mqtt_example (unified-surf-238118)$ gcloud pubsub subscriptions
pull --auto-ack projects/unified-surf-238118/subscriptions/my-subscription
┌──────────────────────────────────┬─────────────────┬──────────────────────────────────────┐
│               DATA               │    MESSAGE_ID   │              ATTRIBUTES              │
├──────────────────────────────────┼─────────────────┼──────────────────────────────────────┤
│ my-registry/my-node-device-payload-3 │ 520858070376222 │ deviceId=my-node-device              │
│                                  │                 │ deviceNumId=2630202724305004         │
│                                  │                 │ deviceRegistryId=my-registry         │
│                                  │                 │ deviceRegistryLocation=us-central1   │
│                                  │                 │ projectId=unified-surf-238118        │
│                                  │                 │ subFolder=                           │
└──────────────────────────────────┴─────────────────┴──────────────────────────────────────┘
vit_solcik@cloudshell:~/nodejs-docs-samples/iot/mqtt_example (unified-surf-238118)$
```

Obrázek 2.8: **Vyzvednutí zpráv daného subscribtion v Google Cloud Shell**

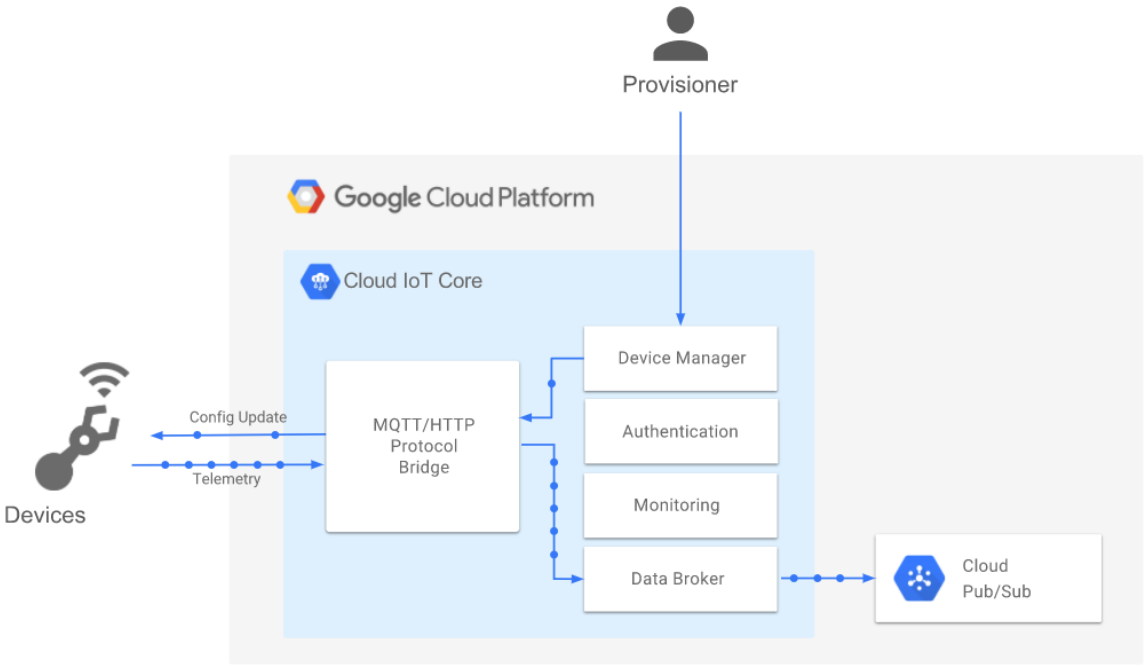
Z výpisu výše je patrné, že zprávy i nastavení komunikace na straně serveru je možno bez potíží zvládnout v pár krocích díky dobře propracovaným materiálům ze strany Google. Stejně jako je tomu u řešení od IBM, Google Cloud IoT zvládá komunikaci pomocí jak MQTT, tak také HTTPS protokolu. Bohužel všechny připravené aplikace obsažené v SDK [12] předpokládají, že cílové zařízení je v souladu s POSIX (Portable Operating System Interface), což většina IoT vestavěných zařízení není. Google nicméně poskytuje portovací dokument, který definuje moduly nutné přepsat za účelem přenesení technologie na jinou platformu zařízení (Board Support Package) [13]:

- BSP IO NET: integrace síťového stacku
- BSP TLS: Transport Layer Security – integrace kryptografických protokolů

  V současnosti je podporována komerční WolfSSL knihovna a volně přístupná knihovna MbedTLS.
- BSP MEM: správa paměťové haldy (heap)
- BSP RNG: generátor náhodných čísel

- BSP CRYPTO: ECC, SHA256, Base64
- BSP TIME: funkce reálného času

Obrázek níže 2.9 nám znázorňuje hierarchii komunikace, která je do značné míry podobná schématu, o němž padla zmínka v souvislosti s AWS. Za povšimnutí stojí dvojí sada protokolového vybavení, které je oproti zmíněnému řešení od AWS obohaceno o podporu HTTPS.

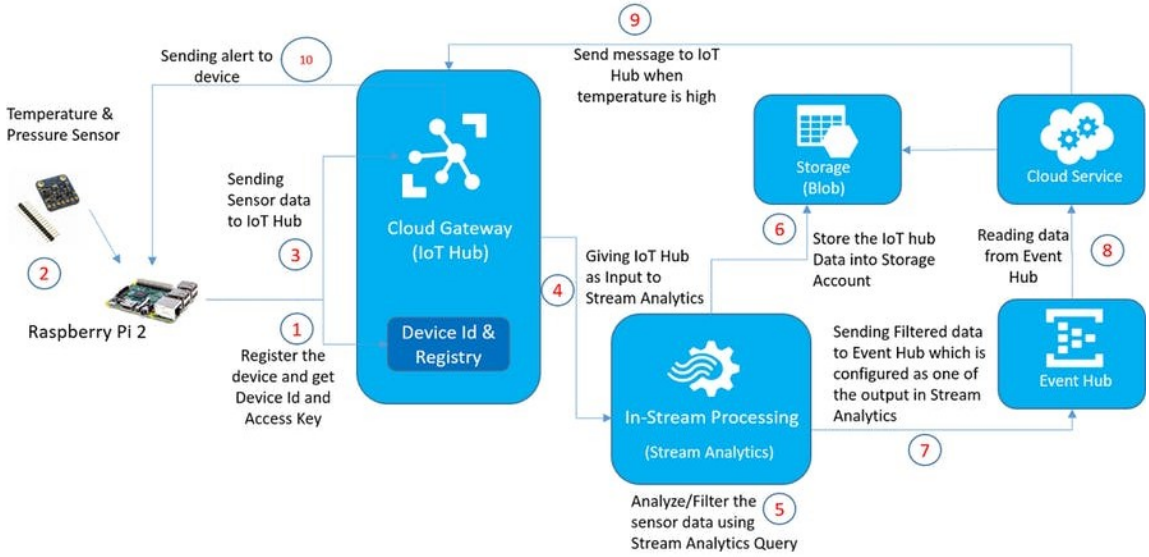


Obrázek 2.9: **Architektura komunikace Google Cloud IoT.** Diagram shrnuje komponenty služby a tok dat [11].

2.4 Microsoft Azure

Poslední ze zmíněných a zároveň komerčně rozšířených platforem popsaných v této práci je Azure od společnosti Microsoft, který se může pyšnit poměrně dobře zpracovanou podporou a obsáhlým SDK, což velmi usnadňuje vývoj řešení postavených na této koncepci. Serverová strana se asi nejvíce vymyká konceptu doposud probraných řešení. Samotné zařízení se připojí k instanci takzvaného IoT Hubu, ve které je zapotřebí vytvořit identitu zařízení, což doprovází vygenerování unikátního řetězce připojení (connection string), díky kterému se v aplikaci na vestavěném zařízení vygeneruje token potřebný pro zabezpečenou komunikaci.

IoT Hub sice umí zprávy přijímat, nicméně sám o sobě nedisponuje mechanismem na zpracování nebo směrování zpráv (pokud pomineme pouhé odeslání potvrzení přijetí zprávy na IoT Hubu). Tato skutečnost nutí uživatele k zakoupení dalších služeb, jako je Event Hub nebo Azure Funcions [23], které již oproti IoT Hubu nenabízí bezplatnou verzi. Poněkud složitější infrastruktura odeslání, zpracování a odpovědi na zprávu je znázorněna na obrázku 2.10.



Obrázek 2.10: **Azure infrastruktura.** Diagram posloupnosti zpracování zpráv na Microsoft Azure platformě [10].

Microsoft Azure nabízí největší počet síťových protokolů mezi doposud diskutovanými platformami a jejich mutací, skrze které s IoT platformou lze komunikovat. Konkrétně se jedná o tyto protokoly:

- MQTT
- MQTT-WS
- AMQP
- AMQP-WS
- HTTPS

SDK pro připojení zařízení v jazyce C je poměrně komplexní. Podporován je klasicky běh aplikace pod systémy Linux, Windows a MBED. Oproti konkurenci však přináší podporu nízkoúrovňového programátorského přístupu na součástce ESP8266 bez použití systémových knihoven. Součástí repozitáře je taktéž portovací dokumentace, která sestává z následujících kroků:

- agenttime adapter - adaptér funkcí reálného času
- sleep adapter - adaptér uspání vlákna závislý na ThreadAPI
- platform adapter - adaptér obsahující inicializaci a deinicializaci platformy
- threadapi and lock adapters - adaptér správy vláken a synchronizačního paradigma (tento adaptér je volitelný)
- tlsio adapter - adaptér zabezpečené komunikace modelu TCP/IP
- socketio adapter - adaptér zřetězení adaptéru komunikace (volitelný)

2.5 Porovnání platforem

Přestože si klade platforma každého z dodavatelů technologie pro IoT stejné cíle, jejich přístup a provedení se pochopitelně liší. Následující podkapitola shrnuje několik postřehů, kde se platformy rozcházejí.

Komunikační protokoly

Asi největším rozdílem z pohledu implementace konektivity na vestavěném zařízení se jeví použití komunikačního síťového protokolu, tabulka 2.1 toto shrnuje.

	AWS	**IBM Watson**	**Google Cloud IoT**	**MS Azure**
HTTPS	X	✓	✓	✓
MQTT	✓	✓	✓	✓
AMQP	X	X	X	✓

Tabulka 2.1: **Porovnání protokolů.** Tabulka zobrazující jednotlivé platformy a jimi podporované komunikační protokoly.

V tabulce je možné vidět, že nejrozšířenějším protokolem je bezesporu MQTT. Na druhou stranu protokol AMQP má zastoupení pouze u Microsoftu, což je více než očekávané vezmeme-li v potaz, že právě Microsoft je jeden z hlavních přispěvatelů při tvorbě tohoto protokolu.

MQTT je standardní publish/subscribe protokol s častým užitím na vestavěných zařízeních a komunikaci mezi nimi. HTTP je protokol bez navázaného spojení (connectionless), tudíž zařízení neudržuje spojení, ale komunikuje modelem dotaz a odpověď.

Vlastnosti protokolu MQTT:

- Nižší využití šířky pásma
- Nižší latence
- Vyšší propustnost
- Podporuje surová binární data

Vlastnosti protokolu HTTP:

- Lehčí nasazení (snadné spuštění, jednoduché příkazy)
- Méně problémů s bránou firewall
- Binární data musí být zakódována base64, což vyžaduje více zdrojů sítě a CPU

Cenová politika

Dobrou zprávou je, že každá z platforem nabízí nějakou možnost (alespoň po omezenou dobu) bezplatného využití jejich služeb. Následuje shrnutí faktorů omezujících takové užití.
AWS [2]:

- 2,250,000 minut připojení všech zařízení
- 500,000 zpráv
- 225,000 operací nad registry zařízení
- 250,000 spuštění pravidel a 250,000 vykonaných akcí
- Celkové časové omezení na 12 měsíců

IBM [18]:

- Maximum 500 registrovaných zařízení
- Maximum 500 aplikačních vazeb (bindings)
- Maximum 200MB přenesených i analyzovaných dat
- Po uplynutí 30-ti denní neaktivity je služba smazána

Google Cloud IoT [14]:

- Maximálně 250 MB přenesených zpráv (na každou zprávu se hledí, že má minimální délku 1024B).

Microsoft Azure [19]:

- Maximum 8,000 zpráv za den
- Maximální velikost zprávy 0,5kB

Porovnání využitelnosti pro vestavěné zařízení

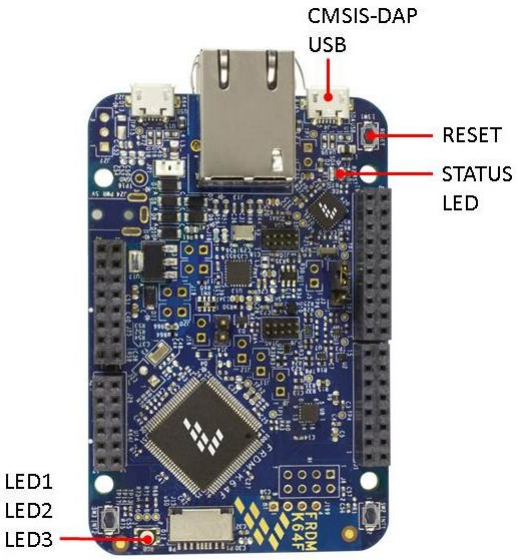
Je více než očekávané, že řešení každého výrobce je závislé na některém z druhů systémové výbavy (časovače, sockety, POSIX, ...), jelikož je implementace na konkrétní vestavěné zařízení úzce spjata s HW, který má konkrétní zařízení k dispozici. Na základě této skutečnosti je od zákazníka očekávána jistá práce za účelem přenesení řešení na jeho platformu.

Porovnáme-li zmíněné platformy výrobců v této práci z pohledu vestavěných zařízení, jsou pro tyto účely patrně nejvhodnější podklady od společnosti Microsoft a jeho platformu Azure. Má nejrozsáhlejší výběr implementací na různé platformy, jejichž portování je kvalitně zdokumentováno a doprovázeno demonstračními aplikacemi. Z této skutečnosti se nadále bude tato práce zabývat zmíněnou platformou Azure. Faktem ale zůstává, že většina dosažených výsledků na straně vestavěného zařízení jsou znovu využitelné při realizaci připojení na všechny ostatní platformy.

Pro demonstraci dosažených výsledků této práce byla zvolena vývojová deska FRDM-K64F (obr. 2.11) od výrobce NXP Semiconductors. Hlavním důvodem výběru této desky byla HW a SW výbava, která splňuje požadavky definované Microsoftem pro komunikaci s Azure IoT Hubem:

- Schopnost navázat spojení IP: pouze zařízení s podporou protokolu IP mohou komunikovat přímo s modulem Azure IoT Hub.
- Podpora TLS: k vytvoření bezpečného komunikačního kanálu s Azure IoT Hub.
- Podpora SHA-256 (volitelné): nutné pro generování zabezpečeného tokenu pro ověření zařízení se službou. K dispozici jsou různé metody ověřování a ne všechny vyžadují SHA-256.
- Hodiny reálného času nebo implementovat kód pro připojení k serveru NTP: nutné pro vytvoření připojení TLS a generování zabezpečeného tokenu pro ověřování.
- Minimálně 64 kB paměti RAM: ovlivněno zvoleným komunikačním protokolem.
- Minimálně 4 kB paměti RAM určených pro příchozí SSL buffer: Během TLS handshake služba IoT Hub pošle Server Hello paket, který obsahuje serverové certifikáty. Při ověření těchto certifikátů se na straně serveru provede kontrola, aby se zabránilo tomu, že Server Hello paket překročí limit 4kB.

Další z důvodů byla oblíbenost a velká rozšířenost této vývojové desky mezi spotřebiteli i vývojáři.



Obrázek 2.11: **Vývojový deska FRDM-K64F**[6].

Kapitola 3

TCP/IP Stack a komunikace

Využití systémových knihoven a socketů by bylo přinejmenším pohodlné a příjemné. Nicméně z důvodů omezených zdrojů a snaze o minimalismus na straně vestavěných zařízení, je tato možnost pro účely práce nevyhovující. Tudíž je potřeba najít vhodnou implementaci, která tyto požadavky splní.

3.1 Lightweight TCP/IP stack (lwIP)

Jako vhodnou variantu pro TPC/IP stack byla zvolena knihovna lwIP (Lightweight TCP/IP stack), která podporuje dostatečnou škálu protokolů a je dnes hojně využívána velkým zastoupením výrobců vestavěných zařízení.

Obecné specifikace a vznik

lwIP je malá nezávislá implementace sady protokolů TCP/IP.

Implementace protokolu lwIP TCP/IP se zaměřuje na snížení využití paměti RAM při plném pokrytí rozsahu TCP. Tím je lwIP vhodný pro použití ve vestavěných systémech s desítkami kB volné paměti RAM a přibližně 40 kB kódu v paměti ROM.

LwIP byl původně vyvinut Adamem Dunkelsem v laboratořích Počítačových a Síťových Architektur ve Švédském Institutu pro Počítačové Vědy (Computer and Networks Architectures lab at the Swedish Institute of Computer Science) a je nyní vyvíjen a udržován celosvětovou sítí vývojářů [1]. Je volně dostupný (pod licencí BSD) ve formátu zdrojového kódu C a lze jej stáhnout z domovské stránky vývoje [9].
Některé klíčové funkce nabízené lwIP [1]:

- IP (Internet Protocol, IPv4 a IPv6) včetně přesměrování paketů skrze více síťových rozhraní
- ICMP (Internet Control Message Protocol) za účelem síťové správy a debuggování
- DHCP, AutoIP/APIPA (Zeroconf) a (bezstavové) DHCPv6
- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- Volitelné API socketů typu Berkeley
- DNS (Domain name resolver včetně mDNS)

Použití lwIP DHCP

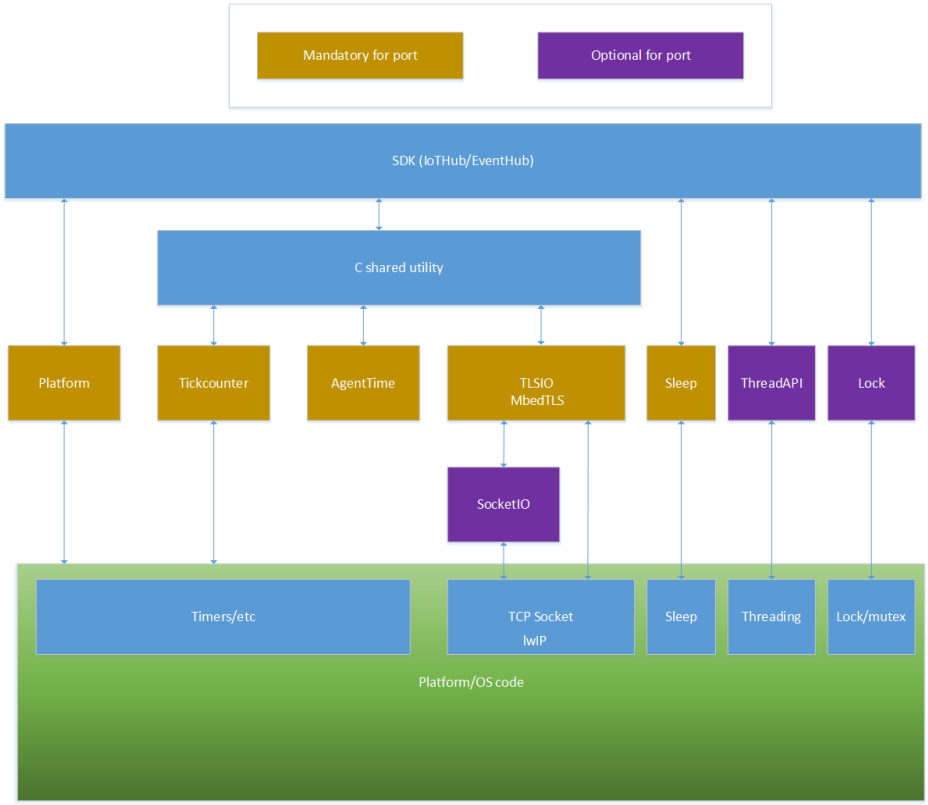
Přestože je možné nastavit konfiguraci sítě (IP adresa, výstupní brána, maska sítě) staticky a během vývoje byl tento postup realizován, od běžného uživatele není tento přístup očekáván. Z toho důvodu je využita implementace komunikace s DHCP serverem skrze knihovnu lwIP, která vyšle všesměrově DHCP discover packet a dotáže se DHCP serveru pro konfiguraci a případné zapůjčení adresy s dalšími konfiguracemi nutnými pro správné připojení k síti.

Použití lwIP SNTP

Hodiny reálného času, které jsou nutné kvůli vytvoření připojení TLS a generování zabezpečeného tokenu pro ověřování během připojování k Azure IoT Hub, jsou realizované připojením k serveru SNTP. Pro tyto účely je přepsán adaptér obsažený v souboru `sntp_lwip.c`, kde bylo potřeba pár minoritních úprav oproti verzi poskytnuté v Azure IoT C SDK z důvodu využití systémové knihovny `time.h`. Tato knihovna má další návaznost na použité vývojové prostředí (je podporováno: IAR Embedded Workbench, Keil® MDK , MCUXpresso IDE a ARMGCC), což je ošetřeno v nově vytvořeném adaptéru `agenttime_msdk.c`.

Použití lwIP socketu s Azure IoT

Tato podkapitola se zabývá napojením lwIP knihovny na síťové adaptéry knihovny Azure IoT Hub. Na obrázku 3.1 lze vidět celkové schéma adaptérů a jejich vzájemné napojení.



Obrázek 3.1: **Azure portační adaptéry.** Diagram návaznosti a propojení jednotlivých adaptérů celého systému. [20].

Po prvotní analýze existujících řešení se jeví jako nejlépe využitelná a dlouhodobě udržitelná varianta Berkley socketů (BSD sockets), kde se veškeré volání systémových socketů pomocí definic přemostí na volání socketů knihovny lwIP. V úryvku kódu můžeme takové volaní vidět. Konkrétně se jedná o navázání spojení se serverem.

```
if (socket_io_instance->address_type == ADDRESS_TYPE_IP)
{
    char portString[16];

    memset(&addrInfoHintIp, 0, sizeof(addrInfoHintIp));
    addrInfoHintIp.ai_family = AF_INET;
    addrInfoHintIp.ai_socktype = SOCK_STREAM;

    sprintf(portString, "%u", socket_io_instance->port);
    err = getaddrinfo( socket_io_instance->hostname,
                       portString, &addrInfoHintIp, &addrInfoIp);
    if (err != 0)
    {
        LogError("Failure: getaddrinfo failure %d.", err);
        result = __FAILURE__;
    }
    else
    {
        connect_addr = addrInfoIp->ai_addr;
        connect_addr_len = sizeof(*addrInfoIp->ai_addr);
        result = 0;
    }
}

if (result == 0)
{
    err = connect(socket_io_instance->socket, connect_addr, connect_addr_len);
    if ((err != 0) && (errno != EINPROGRESS))
    {
        LogError("Failure: connect failure %d.", errno);
        result = __FAILURE__;
    }
}
```

Funkce `getaddrinfo` a `connect` jsou preprocesorovou direktivou definice přetíženy na volání funkcí knihovny lwIP, kde se nachází i deklarace struktur potřebných jako parametry těchto funkcí:

```
#define gethostbyname(name) lwip_gethostbyname(name)
#define connect(s,name,namelen) lwip_connect(s,name,namelen)
```

Další funkce potřebné pro správné fungování síťové TCP/IP komunikace jsou řešeny analogicky

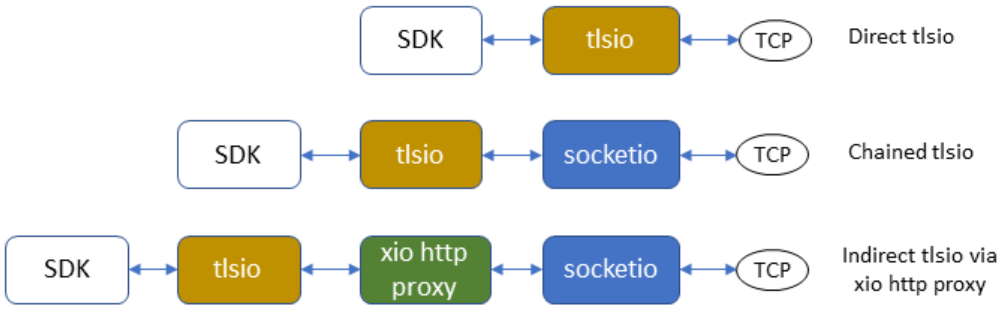
Kapitola 4

Zabezpečená komunikace TLS

Adaptér TLSIO (viz obr. 3.1) zprostředkovává zabezpečenou TLS (Transport Layer Security) komunikaci s Azure IoT za využití zdrojů dosažených a popsaných v předchozí kapitole 3. Implementace adaptéru připouští dva různé přístupy: přímý (direct) a zřetězený (chained). V případě přímého přístupu adaptér vlastní TCP socket, který používá k přímé komunikaci typu TLS se vzdáleným hostem. Naopak v případě zřetězeného stylu adaptér nevlastní TCP socket a se vzdáleným hostem komunikuje nepřímo. Adaptér TLSIO stále provádí všechny úkony spojené s TLS komunikací jako je šifrování, dešifrování a vyjednávání komunikace. Nicméně komunikuje se vzdáleným hostitelem skrze další adaptér, který představuje další abstrakční vrstvu. V našem případě se tento adaptér označuje jako socketio.

Přímý styl je implementačně méně náročný a eliminuje vícero kopírování bufferu, tudíž je i z hlediska paměťové a výkonové náročnosti příznivější. Na druhou stranu zřetězený přístup, přestože je více náročný na zdroje, nabízí vetší flexibilitu. V důsledku této skutečnosti docílíme menší závislosti rozdílných implementací návazných adaptérů a případných HW změn ovlivňující tuto implementaci (například při portování řešení na jinou vývojovou desku/procesor) [20].

Výběr knihovny zaštiťující TLS komunikaci může přímo ovlivnit jaký styl adaptéru TLSIO bude použit. Například implementace podporující Arduino podporuje pouze přímé spojení. Grafické znázornění různých adaptérů můžeme vidět na obr. 4.1.



Obrázek 4.1: **TLSIO Adaptér.** Diagram různých implementací TCP/IP s využitím TLS [20].

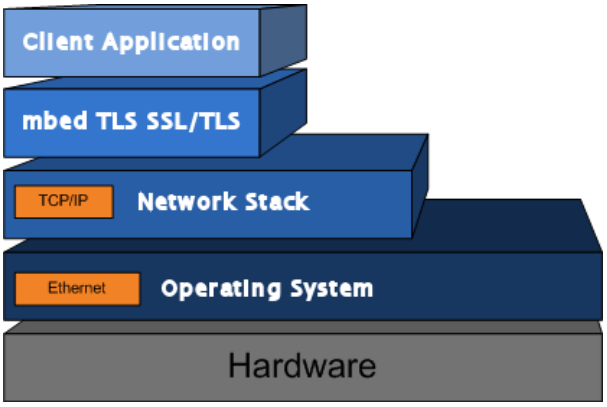
4.1 MbedTLS

Knihovna mbedTLS byla navržena tak, aby se snadno integrovala s existujícími aplikacemi se zaměřením na vestavěné systémy, zatímco poskytuje stavební bloky pro bezpečnou komunikaci, kryptografii a správu klíčů.

S ohlédnutím na specifikaci popsanou v předchozích odstavcích, byla pro vývoj zvolena právě tato kryptografická knihovna. MbedTLS umožňuje obě varianty adaptéru TLSIO zmíněných v předchozí podkapitole, ze kterých byla pro výsledné řešení zvolena jeho komplexnější zřetězená varianta.

Knihovna je navržena tak, aby se daly integrovat pouze ty části, které potřebujeme. Tato skutečnost má za následek velmi nízkou paměťovou náročnost. Odstraněním částí, které nepotřebujete ve svém systému, lze docílit velikosti od 45kB až po typičtějších 300 kB pro plně funkční nastavení. Knihovna MbedTLS byla navržena v jazyce C a je distribuována pod licencí Apache 2.0.

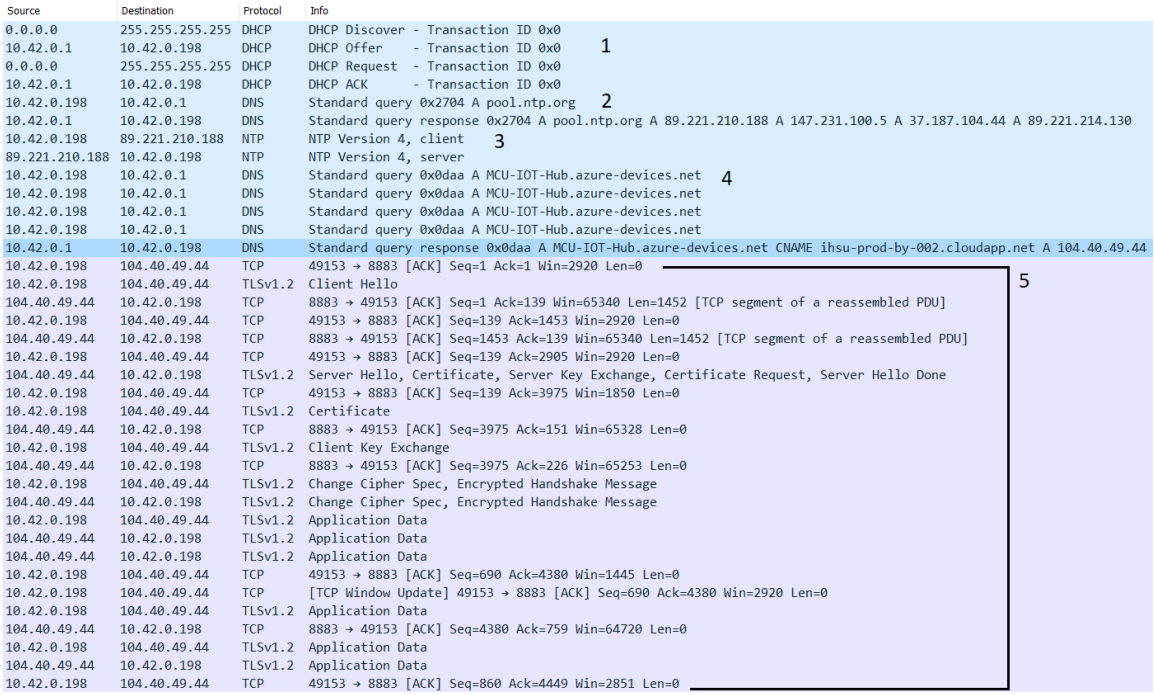
Na obrázku 4.2 můžeme vidět strukturu celé aplikace z pohledu zmíněné knihovny.



Obrázek 4.2: **Knihovna mbedTLS.** Znázornění kryptografické vrstvy v aplikaci [8].

Za zmínku také stojí, že knihovna podporuje využití kryptografického akcelerátoru MMCAU (Memory Mapped Arithmetic Unit), který urychluje sadu matematických operací včetně znaménkového i bez znaménkového násobení, sčítání, dělení, odmocnění, a dalších operací potřebných v prostředí kryptografie.

Celá zabezpečená komunikace byla pomocí přepínače se zrcadleným portem a síťovou sondou prostředí software Wireshark zachycena a analyzována (pozn. další metodou je sdílené internetové připojení na vestavěné zařízení skrze počítač se síťovým analyzátorem). Získání těchto dat usnadnilo vývoj a nastínilo přibližný obraz celé komunikace, jelikož samotná užitečná data byla již šifrována. Rozbor této komunikace lze vidět na obrázku 4.3 a celá komunikace je k dispozici na kompaktním disku (viz. příloha B).



Obrázek 4.3: **Ukázka komunikace zachycená nástrojem Wireshark.** Zobrazení zachycených paketů při komunikaci vestavěného zařízení a IoT Hubu. 1) Dotaz, nabídka a přijetí síťové konfigurace z DHCP serveru. 2) DNS dotaz a odpověď na adresu serveru reálného času služby NTP. 3) NTP přenos. 4) Dotaz na DNS překlad adresy serveru Microsoft Azure získané z connection stringu. 5) Navázání šifrované TLS komunikace sestávající z dohodnutí verze protokolu, výběr kryptografického algoritmu, výměna a zkontrolování certifikátů, vygenerování sdíleného tajného klíče pomocí asymetrického šifrování a následná rychlejší symetricky šifrovaná výměna samotných dat.

Kapitola 5

Knihovna Microsoft Azure

Samotná knihovna (chcete-li SDK) Azure po úspěšném implementování dříve popsaných adaptérů nevyžadovala vetší zásahy, což je dobrou vizitkou kódů poskytnutých od společnosti Microsoft a tato skutečnost usnadní další údržbu celé platformy na straně výrobce procesorů. Změny nutné pro správné fungování a zamezení kompilačních chyb jsou popsány v changelogu ve složce knihovny a sestávají hlavně z těchto bodů:

- Nové soubory:
  - c-utility\adapters\agenttime_msdk.c - přidání adaptační vrstvy reálného času
  - c-utility\adapters\platform_msdk.c - přidání adaptační vrstvy platformy
  - c-utility\adapters\socketio_berkeley_msdk.c - přidání lwIP adaptační vrstvy
- Odstraněné soubory:
  - azure_c_shared_utility\tls_config.h - zastaralý konfigurační soubor MbedTLS
- Přejmenované soubory (Oprava konfliktů se systémovými knihovnami nebo jiným firmware):
  - c-utility\src\base64_azure.c
  - c-utility\src\hmac_azure.c
  - c-utility\src\sha1_azure.c
  - iothub_client\src\version_azure.c
- Změněné soubory:
  - c-utility\adapters\tlsio_mbedtls.c
    - odstraněno zastaralé USE_MBED_TLS podmínkové macro
    - include string.h, stdio.h, fsl_debug_console.h
    - zakomentování MBED_TLS_DEBUG_ENABLE definince
    - zakomentování funkcí mbedtls_ssl_conf_dbg a mbedtls_debug_set_threshold
  - c-utility\inc\azure_c_shared_utility\agenttime.h - přidána deklarace funkce sntp_get_current_timestamp
  - c-utility\inc\azure_c_shared_utility\crt_abstractions.h - Přidána ENOMEM a EINVAL definice (IAR toolchain specifické)

- c-utility\inc\azure_c_shared_utility\gbnetwork.h - include lwip/sockets.h
- c-utility\inc\azure_c_shared_utility\tlsio_mbedtls.h
  - odstraněno zastaralé USE_MBED_TLS podmínkové macro
  - odstraněno linkování zastaralého konfiguračního souboru tls_config.h
- c-utility\pal\freertos\lock.c - upravení FreeRTOS cesty z důsledku změny hierarchie složek
- c-utility\pal\freertos\threadapi.c
  - upravení FreeRTOS cesty
  - použití definice FreeRTOSu portTICK_PERIOD_MS, která je závislá na použité architektuře
- c-utility\pal\freertos\tickcounter.c - použití definice FreeRTOSu portTICK_PERIOD_MS, která je závislá na použité architektuře
- c-utility\pal\lwip\sntp_lwip.c
  - include time.h
  - rozdílná implementace SNTP_Inits (využití agenttime_msdk.c adaptéru)
- deps\parson\parson.c - vyloučení funkcí pracujících s file systémem (externí JSON knihovna)
- iothub_client\src\iothub_client_diagnostic.c - odstranění int64_t kontroly z důvodu absence tohoto typu (ARMGCC)

Celý popis hierarchie knihovny s popisem jednotlivých částí je k nahlédnutí v příloze A.

Aplikační protokoly

Díky poměrně vysoké flexibilitě (na poměry vestavěných systému) a použití abstrakčních vrstev je možné použít různé aplikační protokoly. Microsoft Azure na platformě napsané v jazyku C podporuje HTTPS, MQTT a AMQP. V rámci této práce bylo realizováno připojení pomocí protokolů HTTPS a MQTT, nicméně ne souběžně. Každý z využitých protokolů disponuje vlastní aplikací, které se mezi sebou liší použitím jiných souborů implementující komunikaci, jenž se do ní linkují. Jejich výčet je možné vidět v tabulce 5.1.

HTTPS	**MQTT**
mqtt_client.c/.h mqtt_codec.c/.h mqtt_message.c/.h mqttconst.h	iothubtransporthttp.c iothubtransporthttp.h

Tabulka 5.1: **Výčet souborů různých protokolů.**

Příprava vývojové desky

Aplikace na vestavěném zařízení musí obsahovat ve zdrojovém kódu tuto globální proměnnou (jejíž jednotlivé části jsou vysvětleny v komentáři):

```
/*String containing Hostname, Device Id & Device Key in the format:
 "HostName=<host_name>;DeviceId=<device_id>;SharedAccessKey=<device_key>"
 "HostName=<host_name>;DeviceId=<device_id>;
 SharedAccessSignature=<device_sas_token>" */
static const char* connectionString = "[device connection string]";
```

Jak již bylo zmíněno v podkapitole 2.4, tento řetězec připojení (connection string) je vygenerován na straně serveru při vytvoření identity zařízení v rámci dané instance IoT Hubu. Po zadání tohoto řetězce do stejně pojmenované proměnné a zkompilování kódu je vše připravené pro nahrání aplikace na vývojovou desku. Postup samotného nahrání je závislý na použitém vývojovém prostředí, debuggovacím rozhraní, firmware debuggeru a v neposlední řadě na druhu desky a její verzi. Všechny tyto aspekty jsou vzaty v potaz a popsány v dokumentaci příslušné desky. V případě vývojové desky K64F použité pro účely demonstrace výsledků se dokumentace nachází na tomto odkazu[1]. Po úspěšném nahrání aplikace je nutné nastavit komunikaci přes UART rozhraní s PC pro příjem výstupu z desky do konzole (propojení USB kabelem mezi hostitelským PC a OpenSDA USB nebo USB a sériové linky rozhraní s deskou předpokládáme již pro účely nahrání aplikace). Typicky je použito toto nastavení komunikace UART:

- COM port (lze zjistit z manažeru zařízení na PC; podrobnější popis v dokumentaci výrobce desky)
- baud rate 115 200
- No parity
- 8 data bity
- 1 stop bit

Posledním krokem před samotným spuštěním aplikace je nutné připojit desku ethernetovým kabelem (port RJ45) do internetové sítě disponující DHCP serverem a konfigurací dovolující komunikaci s DNS, NTP, MQTT/HTTP službami. Porty některých z těchto služeb mohou být někdy blokovány firewallem:

- DNS: 53
- NTP: 123
- MQTT: 8883 (přes SSL)
- HTTPS: 443

[1] https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/k-seriesperformancem4/k6x-ethernet/kinetis-k64-120-mhz-256kb-sram-microcontrollers-mcus-based-on-arm-cortex-m4-core:K64_120

Kapitola 6

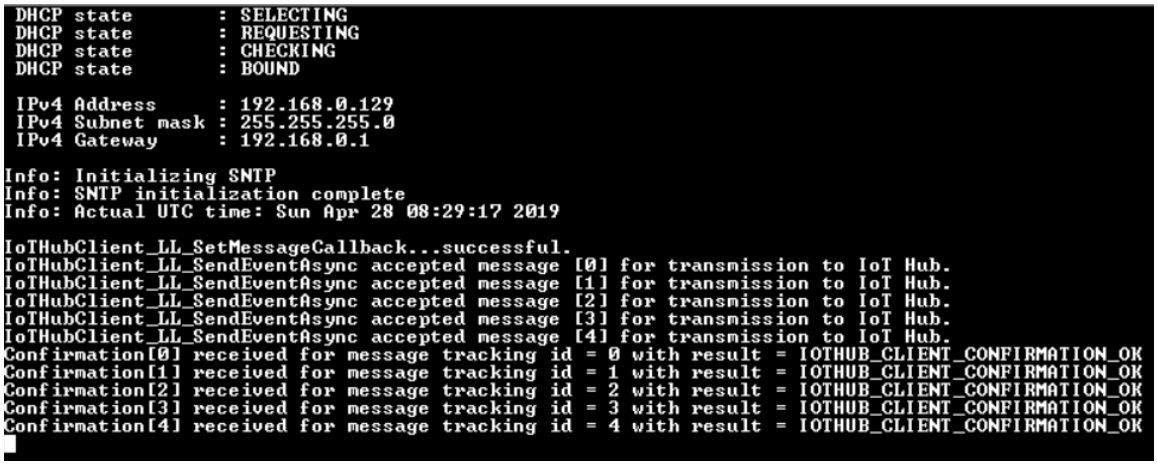
Testování a demonstrace výsledků

Tato kapitola pojednává o využití dosažených cílů a předpokládá, že zařízení je připraveno podle kroků popsaných v předchozí kapitole 5, zatímco na straně PC mámě otevřenou konzolu sériové linky pro jejich zobrazení.

Skutečnost, že zprávy opravdu doputují ke svému cíli, může být otestována jednoduše pomocí počítadla přijatých zpráv na serverové straně Azure portálu nebo přijetím feedback (zpětnovazební) zprávy ze serveru potvrzující příjem předtím odeslané zprávy. Výstup takovéto komunikace za použití aplikačního protokolu HTTPS lze vidět na obrázku 6.1 a za použití MQTT na obrázku 6.2.

```
 DHCP state       : SELECTING
 DHCP state       : REQUESTING
 DHCP state       : CHECKING
 DHCP state       : BOUND

 IPv4 Address     : 192.168.0.129
 IPv4 Subnet mask : 255.255.255.0
 IPv4 Gateway     : 192.168.0.1

Info: Initializing SNTP
Info: SNTP initialization complete
Info: Actual UTC time: Sun Apr 28 08:26:46 2019

Starting the IoTHub client sample HTTP...
IoTHubClient_LL_SetMessageCallback...successful.
IoTHubClient_LL_SendEventAsync accepted message [0] for transmission to IoT Hub.
Confirmation[0] received for message tracking id = 0 with result = IOTHUB_CLIENT_CONFIRMATION_OK
IoTHubClient_LL_SendEventAsync accepted message [1] for transmission to IoT Hub.
Confirmation[1] received for message tracking id = 1 with result = IOTHUB_CLIENT_CONFIRMATION_OK
IoTHubClient_LL_SendEventAsync accepted message [2] for transmission to IoT Hub.
Confirmation[2] received for message tracking id = 2 with result = IOTHUB_CLIENT_CONFIRMATION_OK
IoTHubClient_LL_SendEventAsync accepted message [3] for transmission to IoT Hub.
Confirmation[3] received for message tracking id = 3 with result = IOTHUB_CLIENT_CONFIRMATION_OK
IoTHubClient_LL_SendEventAsync accepted message [4] for transmission to IoT Hub.
Confirmation[4] received for message tracking id = 4 with result = IOTHUB_CLIENT_CONFIRMATION_OK
```

Obrázek 6.1: **Komunikace s Azure IoT Hub skrze HTTPS**



Obrázek 6.2: **Komunikace s Azure IoT Hub skrze MQTT**

Jak jste si mohli všimnout, hlavní rozdíl ve výpisu HTTPS a MQTT aplikace je pořadí odeslání zpráv a příjem zpětné vazby ze serveru s potvrzením přijetí zprávy. To lze zdůvodnit faktem, že HTTPS komunikace je postavena na request-response modelu (dotaz-odpověď), zatímco protokol MQTT na modelu publish-subscribe (publikování-odběr).

Servisní aplikace

Za účelem lepší demonstrace dosažených výsledků byla vyvinuta snaha zprávy poslané na server nějakým způsobem zpracovat a poslat na zařízení připojené k IoT Hubu. Nicméně tento model komunikace je dosažitelný pouze skrze nasazení a napojení dalších služeb Microsoft Azure (Event Hub, Functions), které jsou zpoplatněné. Pro demonstrační účely jsou tyto metody zbytečně těžkopádné a jejich využití je mimo rozsah této práce.

Z těchto důvodů byl zvolen jiný přístup využívající servisních knihoven Azure, které se nechovají jako samotné zařízení, jelikož nemají ani svojí identitu v registru zařízení. Výhodou je, že umožňují zasílání cloud-to-device zpráv, jichž je využito. Takováto komunikace přináší omezení, že je komunikace jednosměrná (unidirectional), jelikož zmíněná servisní aplikace nemá identifikátor, na který by se druhá strana mohla odkazovat a posílat zprávy. Nicméně i servisní aplikace disponuje mechanismem pro příjem zpětnovazebního potvrzení, které je serverem vygenerováno pouze v případě, že zpráva úspěšně dorazila na cílové (vestavěné) zařízení. Toto zařízení na své straně po přijetí zprávy vždy informuje server o jejím obdržení.

Knihovna Microsoft Azure nepodporuje servisní přístup v jazyce C (pozn.: v době vývoje nepodporovala, nyní v omezeném rozsahu pouze AMQP protokol), tudíž je zvolena knihovna v jazyce Java. Byla vytvořena desktopová aplikace, která naváže spojení s IoT Hubem a posílá cloud-to-device zprávy, které mají podobu příkazu pro rozsvícení a zhasnutí LED na vývojové desce (vestavěném zařízení) nebo textové zprávy. Za účelem zpracování těchto zpráv na straně vestavěného systému byly vytvořeny aplikace `remote_control_http` a `remote_control_mqtt`, které jsou mutace předešlých aplikací pro oba z podporovaných protokolů. Aplikace zprávy přijmou a na základě obsahu vyhodnotí jako příkaz rozsvícení/zhasnutí LED nebo jako textovou zprávu, jejíž obsah je přesměrován na výstup.

Oproti běžnému zařízení však nelze pro připojení použít řetězec připojení zařízení, a to jednoduše z důvodu, protože neexistuje (tento řetězec se generuje současně při vytvoření

identity zařízení). Místo toho je použit speciální servisní řetězec, jenž je součástí přístupových práv (access policies) na portálu Azure spravující danou instanci IoT Hubu. Je třeba dát si pozor, aby dané přístupové pravidlo (polici) mělo povolený servisní přístup k IoT Hubu (nejjednodušší je použít genericky vytvořené pravidlo "iothubowner", které má absolutní práva). Dále se použije identifikátor zařízení, se kterým chceme z aplikace komunikovat. Řetězec i identifikátor je nutné zadat přímo ve zdrojovém kódu aplikace v souboru `ServiceClientSample.java`:

```
private static final String connectionString = "[Connection string goes here]";
private static final String deviceId = "[Device name goes here]";
```

Servisní aplikace je zkompilována pomocí nástroje Maven sloužící pro správu, řízení a automatizaci sestavení aplikací. Sestavení aplikace je iniciováno těmito příkazy ve složce aplikace (`service-client-sample`):

```
mvn clean package
mvn clean install
```

Následné spuštění pak příkazem:

```
mvn exec:java -D exec.mainClass=
"samples.com.microsoft.azure.sdk.iot.ServiceClientSample"
```

Jedná se o konzolovou aplikaci, jejíž výstup po úspěšném spuštění, navázání spojení s instancí IoT Hubu a posláním krátké zprávy vypadá takto:

```
$ mvn exec:java -D exec.mainClass="samples.com.microsoft.azure.sdk.iot.ServiceClientSample"
[INFO] Scanning for projects...
[INFO]
[INFO] --< com.microsoft.azure.sdk.iot.samples.service:service-client-sample >--
[INFO] Building Service Client Sample 1.13.3
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ service-client-sample ---
********* Starting ServiceClient sample...
Creating ServiceClient...
********* Successfully created an ServiceClient.
********* Successfully opened FeedbackReceiver...
********* Successfully opened fileUploadNotificationReceiver...
sendMultipleCommandsAndReadFromTheFeedbackReceiver: Send count is : 5

Remote control demo commands:
        1 - LED control
        2 - Message to board
        exit - wait for message feedback
2
Type your message here:Hello Azure IoT Embedded Device!
Message id set: 882378f8-3e7b-4798-8fad-7b18c9e213c1


Remote control demo commands:
        1 - LED control
        2 - Message to board
        exit - wait for message feedback
exit
Waiting for all sends to be completed...
All sends completed !
Waiting for the feedback...
 Feedback received, feedback time: 2019-04-28T14:59:53.956918Z
 Record size: 1
 Messsage id : 882378f8-3e7b-4798-8fad-7b18c9e213c1
 Device id : frdmk64f
 Status description : Success
No file upload notification received !
********* Successfully closed fileUploadNotificationReceiver.
********* Successfully closed ServiceClient.
********* Shutting down ServiceClient sample...
```

Obrázek 6.3: **Servisní aplikace.** Výstup servisní aplikace během komunikace s vestavěným zařízením.

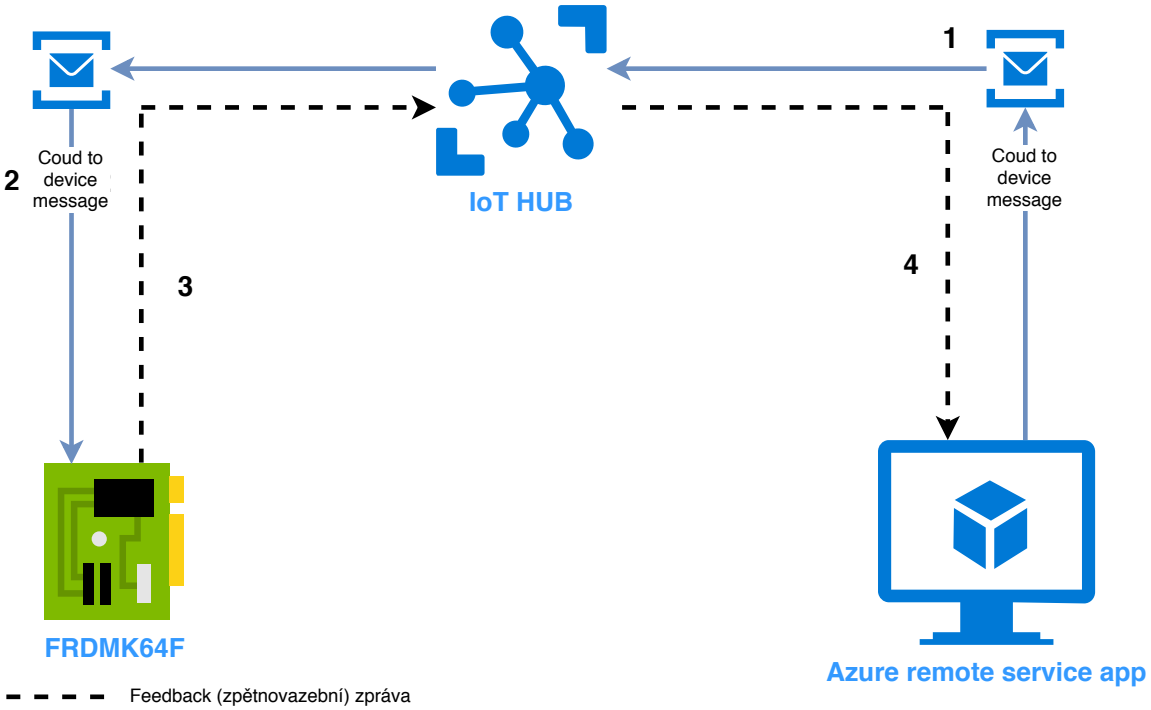
Výstup na straně vestavěného zařízení po odeslání zprávy ze servisní aplikace realizovaného v obrázku 6.3 pak lze vidět na obrázku 6.4:

```
 DHCP state       : SELECTING
 DHCP state       : REQUESTING
 DHCP state       : CHECKING
 DHCP state       : BOUND

 IPv4 Address     : 192.168.0.129
 IPv4 Subnet mask : 255.255.255.0
 IPv4 Gateway     : 192.168.0.1

Info: Initializing SNTP
Info: SNTP initialization complete
Info: Actual UTC time: Sun Apr 28 14:59:20 2019

IoTHubClient_LL_SetMessageCallback...successful.
Recieved message: Hello Azure IoT Embedded Device!

 Message ID: 882378f8-3e7b-4798-8fad-7b18c9e213c1
 Correlation ID: <null>
 Content-Type: <null>
 Content-Encoding: <null>
 Data: <<<Hello Azure IoT Embedded Device!>>> & Size=32
 Message Properties:
        Key: key_0 Value: value_0
```

Obrázek 6.4: **Remote conrol aplikace.** Příjem zpráv ze servisní aplikace na straně vestavěného zařízení.

Komunikace realizována výše je také znázorněna na diagramu 6.5, kde je názorně zobrazena posloupnost jednotlivých akcí. Za předpokladu, že jsou již obě zařízení připojena ke stejné instanci IoT Hubu, servisní aplikace vygeneruje could-to-device zprávu obsahující identifikátor cílového zařízení (1) tuto zprávu IoT Hub příjme a pošle na patřičné zařízení na základě identifikátoru (2). Zařízení zprávu příjme, zpracuje, vyhodnotí akci a následně odešle potvrzení o přijetí zprávy na IoT Hub (3), které obsahuje unikátní identifikátor zprávy, na jehož základě je toto potvrzení propagováno na iniciátora komunikace, v našem případě servisní aplikaci (4).

Z diagramu se může zdát, že komunikace jeví známky obousměrného provozu skrze IoT Hub, nicméně jelikož se jedná pouze o potvrzení, které nenese žádná užitečná data, nelze k této komunikaci takto přistupovat.



Obrázek 6.5: **Diagram komunikace.** Komunikace servisní aplikace s vývojovou deskou FRDMK64F.

Ze skutečnosti, že aplikace na obou stranách přijali a potvrdili komunikaci jak je popsáno výše, lze usoudit správnou funkčnost celé komunikační sestavy a řešení prohlásit za fungující. Další analýza spojení proběhla komunikací přes přepínač se zrcadleným (mirroring) portem, kdy byla komunikace desky se serverem následně zachycena na zmíněném zrcadleném portu pomocí síťové sondy v prostředí software Wireshark. Analýza paketů zprostředkovala další potvrzení správného průběhu komunikace.

Kapitola 7

Další aspekty implementace

Tato kapitola probere některé implementační aspekty, které nebyly doposud popsány, ale jejich zmínění by nemělo být opomenuto.

FreeRTOS

Jak si možná někteří čtenáři této práce povšimli, v kapitole 5 popisující knihovnu Azure je zmíněn FreeRTOS real-time operační systém. Hlavním důvodem bylo užití synchronizace procesů, který tento systém poskytuje. Zejména u navazování zabezpečené komunikace skrze kryptografickou knihovnu, kdy se v průběhu tzv. třícestného handshake předpokládá určitá režie při čekaní odpovědi od serveru. Další ne tak kritický důvod spočívá v tom, že v případě uspání procesu je možné docílit úspornějšího provozu, jenž je u vestavěných systémů (zejména těch napájených z baterií) často podstatným faktorem.

Vývojová prostředí

Ze snahy poskytnout výsledný produkt jako celek co nejširší škále zákazníků, je podporováno hned několik vývojových prostředí (toolchains). Zprvu se tato skutečnost nemusí jevit jako problematická. Nicméně na tak nízké úrovni, jako u vestavěných systémů, může mít změna vývojového prostředí fatální dopad. Hlavní důvody, které tyto neblahé příčiny přináší, je fakt, že každé z prostředí disponuje jiným kompilátorem a systémovými knihovnami, což se problematicky projevilo zejména u implementace adaptéru reálného času. Většinou jsou tyto problémy eliminovány preprocesorovými direktivami, které definují podmíněný překlad některých částí kódu podmínkou definovanou právě používaným prostředím. Z této skutečnosti se někdy může jevit kód redundantní, i když tento fakt neovlivní z výše popsaného důvodu výslednou velikost binárního souboru nahrávaného na cílovou platformu. Podporované vývojové prostředí jsou:

- IAR Embedded Workbench: vývojové prostředí, které obsahuje kompilátor C/C++ a ladicí program (debugger). Podpora vývoje se zaměřením na 8, 16 a 32 bitové mikrokontroléry. Použití vyžaduje zakoupenou licenci (existuje 30-ti denní zkušební verze) [16].
- Keil® MDK: Prostředí obsahuje všechny nástroje potřebné k vývoji aplikací na platformě ARM Cortex-M, což podtrhuje fakt, že společnost Keil byla v roce 2005 odkoupena firmou ARM [5]. Ačkoliv je k dispozici bezplatná verze (s označením Lite), je pro účely této práce nepoužitelná z důvodu omezení velikosti výsledné aplikace na 32kB. Z toho důvodu je také vyžadováno zakoupení licence v rámci tohoto vývoje [7].

- MCUXpresso: Jedná se o vývojové prostředí postavené na populárním prostředí Eclipse. Vývoj je realizován firmou NXP Semiconductors, která je také výrobcem vestavěných zařízení. Tato skutečnost přináší širokou podporu pro vývoj dané platformy (konfigurační nástroje pinů, periferii, časovačů, ...). Jako první ze zmíněných prostředí je jeho použití plně zdarma a poskytuje všechny nástroje potřebné pro vývoj včetně kompilátoru a ladícího programu.

- GNU Arm Embedded: Přestože toto vývojové prostředí nedisponuje grafickým rozhraním, stále se dá do této skupiny zařadit. Jedná se o soubor balíčků s kompilátorem Arm Embedded GCC, knihovnami a dalšími nástroji GNU nezbytnými pro vývoj softwaru na zařízeních s procesorovou architekturou Arm Cortex-M a Cortex-R. Jeho použití je zcela bezplatné a podporuje všechny z rozšířených desktopových operačních systémů. Nicméně je od vývojáře požadována vetší znalost nástrojů a postupů, jelikož se nejedná o jeden celek, ale spíše balíček nástrojů.

Ze stejného důvodu proč bylo podpořeno vícero vývojových prostředí, bylo podpořeno po dokončení prací i více vývojových desek různých řad (Kinetis, LPCxpresso, i.MX RT) výrobce NXP Semuconductors. Tato skutečnost dává zákazníkům širší možnost pro realizaci jejich vlastního řešení a zároveň dokládá poměrně dobrou flexibilitu a přenositelnost celého softwarového řešení. V současné době jsou podporovány tyto vývojové desky:

- FRDM-K64F

  ARM Cortex-M4 32-bit, Floating Point Unit (FPU), 120 MHz CPU, 1024 KB program flash, 256 KB RAM

- FRDM-K66F

  ARM Cortex-M4 32-bit, DSP instrukce, 180 MHz CPU, 2 MB program flash, 256 KB RAM

- LPCXpresso54x018

  Arm Cortex-M4 32-bit, 180 MHz CPU, 4 MB program flash, 360 KB SRAM, (security verze: Physical Unclonable Function (PUF), One-Time Programmable paměť (OTP))

- MIMXRT1050-EVK

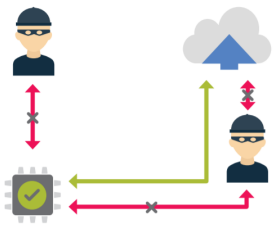
  Arm Cortex-M7 32bit, 600 MHz CPU, 512 MB Hyper Flash, 256 MB SDRAM, 2D grafické akcelerátory, paralelní kamerové senzory, LCD display controller

Kapitola 8

Bezpečnost

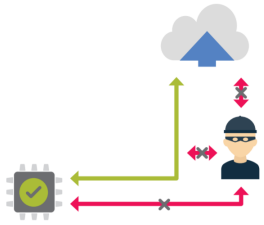
Otázka bezpečnosti je bezesporu jedna z hlavních. Je nejspíše utopie myslet si, že nějaký systém je nenapadnutelný, nicméně je potřeba vyvinout snahu toto riziko minimalizovat a docílit naplnění faktorů ovlivňující zabezpečení:

Integrita
Zajištění nemodifikovaného přenosu dat a provedení nemodifikovaného SW. Obr. 7.1



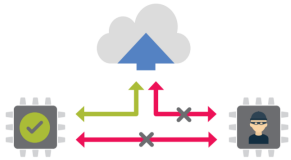
Obrázek 7.1: **Integrita** [21]

Důvěryhodnost
Utajení tajemství a soukromí dat.
Šifrování je zvolenou technologií. Obr. 7.2



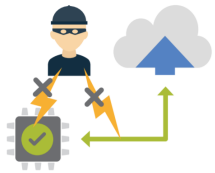
Obrázek 7.2:**Důvěryhodnost** [21]

Autenticita
Ověření identity zdroje dat a SW, řízení přístupu (důvěryhodné operace). Obr. 7.3



Obrázek 7.3: **Autenticita** [21]

Dostupnost
Zajištění dostupnosti služeb. Obr. 7.4



Obrázek 7.4: **Dostupnost** [21]

8.1 Bezpečnostní útoky

Útoky lze klasifikovat podle dvou hlavních charakteristik na vzdálené a lokální nebo fyzické a logické. Následující odstavce jsou citovány z publikace NXP Semiconductors [21].

Lokální útoky jsou prováděny díky fyzickému přístupu k zařízení, zatímco vzdálené útoky jsou prováděny posíláním příkazů vzdáleně skrze síťové připojení. Znalosti nabyté z provedení lokálního útoku mohou v budoucnu vést k realizaci vzdáleného útoku. Ačkoliv vývoj vzdáleného útoku může vyžadovat značné odborné znalosti, je možné tyto útoky automatizovat a vykonat ve značně větší míře. Tato skutečnost naznačuje škálovatelnost vzdálených útoků a fakt, že potencionálně může jedno zařízení ovlivnit milióny zařízení ve velmi krátké době.

Logické útoky na zařízeních, internetových službách nebo organizacích vznikají odhalením slabin v implementaci, které jsou většinou v softwaru. Jsou prováděny přístupem ke standardním rozhraním, jak kabelovým, tak bezdrátovým. Mohou být automatizovány a jakmile jsou odhaleny, nevyžadují markantní znalosti za účelem velkého rozšíření.

Fyzické útoky na zařízení využívají známých nebo naučených fyzikálních vlastností během operací zařízení a prolomení kritického prvku zabezpečení jako jsou kryptografické klíče. Vzdálené fyzické útoky implementované v software, jako je Rowhammer[22], Meltdown/Spectre[15], útoky cílené na vyrovnávací paměť a na kontrolu napájení, se objevily v posledních několika letech.

V tabulce 8.1 lze vidět některé příklady útoků na základě výše popsaného rozdělení.

	Fyzické	**Logické**
Lokální	Analýza výkonu Glitching	Zneužití JTAG, serial, USB rozhraní
Vzdálené	Rowhammer Časování mezipaměti	Přetečení bufferu Heartbleed Zaplavení/DoS

Tabulka 8.1: **Výčet útoků.** Různé útoky z pohledu druhu útoku a názorné příklady jejich konkrétní realizace.

Samozřejmě je vyvíjena snaha a úsilí všechny z hrozeb zmíněných výše eliminovat, nebo alespoň znesnadnit. Základní technikou pro zamezení vzdáleným útokům je bezesporu šifrovaná komunikace. Další podstatná technika je diversita tajných klíčů, která zamezí rozšíření útoku na další zařízení v případě, že se podaří útočníkovi získat tajný klíč jednoho zařízení. Diverzity klíčů mimo jiné lze docílit pomocí periferie PFU (Physical Unclonable Functions), která ze speciální paměti SRAM ihned po startu načte takzvaný aktivační kód, který je určen fyzikálním rozložením polovodiče zmíněné paměti, což zaručuje unikátnost tohoto aktivačního kódu napříč všemi zařízeními a zároveň neměnnost tohoto klíče na daném zařízení. Pomocí aktivačního klíče se následně zašifrují veškeré tajné klíče, kterými dané zařízení disponuje (AES, Firmware update, HMAC klíč, ...).

Zamezení fyzickým útokům je pokryto od základních technik jako je uzavření ladících rozhraní v určitém životním cyklu zařízení (zpravidla po opuštění výrobní linky), tak více sofistikované bezpečnostní čítače instrukcí pro zamezení nechtěného přeskočení instrukcí například metodou glitching, což může vést k nechtěnému vykonání kódu nebo nestandardnímu chování. V neposlední ředě je nutné si uvědomit, že stejně tak jako zamezit samotnému útoku, je důležité případný (i úspěšný) útok detekovat a vyvolat tím techniky minimalizující škody například obnovou firmware.

Další aspekty, které je nutno zvážit v návaznosti na zabezpečení [21]:

- Velké množství stejných zařízení
  - Pokud dojde ke kritickému selhání, je postiženo mnoho zařízeni.
- Velký počet zařízení s jedním sdíleným přístupovým bodem
- Mnoho IoT zařízení je autonomních
  - Zařízení mohou disponovat aktivními prvky jako neurostimulátor. Komunikace s těmito prvky mohou probíhat autonomně bez vědomí uživatele, což vystavuje další možnosti, jak napadnout takovýto systém bez povšimnutí.
- Mnoho IoT zařízení pro mnoho účelů
  - Pro zařízení IoT zatím neexistuje společný standard definující bezpečnostní opatření a zásady jako je tomu například u plateb přes internet a elektronickou identifikaci osob, které jsou centralizovány. Bezpečnost IoT bude vyžadovat úsilí o standardizaci, protože všechna tato zařízení jsou nakonec připojena ke stejné síti. Tyto standardy musí vzít v potaz komplexnost užití IoT v různých odvětvích.
- Doba životnosti IoT zařízení
  - Životnost IoT zařízení není specifikována počtem let jako třeba u platebních karet a cestovních pasů. To znamená, že obsah a zabezpečení zařízení musí být aktualizováno v terénu, aby bylo možné řešit nové útoky nebo chyby zjištěné během jeho životnosti. I v případě, že zařízení je vyvinuto způsobem dovolující jeho aktualizaci, postupem času může být výpočetní výkon nebo paměťová výbava nedostačující a nebo výrobce zařízení může opustit svou činnost dříve, než zařízení, která vyrobil, dosáhla své životnosti. Tím pádem zařízení s různým věkem a s různými úrovněmi zabezpečení budou existovat ve stejném prostředí i systémech. Nahrazení takových zařízení může být obtížné a někdy řešeno přidáním dalšího zařízení zastřešující otázku zabezpečení. I samotná výměna starých zařízení může být výzvou, protože systémy, které kontrolují, musí být navrženy pro takové náhrady. Toto je další důvod pro standardizaci bezpečnostních prvků v IoT.
- Mnoho IoT zařízení má omezené zdroje
  - Mnoho IoT zařízení má omezený výpočetní výkon, kapacitu pamětí a komunikační šířku pásma, což ztěžuje implementaci standardních bezpečnostních technik.

- Zařízení IoT pro kritické a nekritické aplikace jsou spolu v jedné síti
  - Tento fakt poskytuje výhodu útočníkovi, který se může zmocnit kontroly nad slabě zabezpečeným zařízením a využít ho například k DDoS útoku na zabezpečená zařízení, jejichž správné fungování je kritické.
- Zařízení IoT generují obrovské množství osobních dat
  - Zařízení IoT shromažďují pro účely analýzy a strojového učení obrovské množství podrobných dat a posílají informace poskytovatelům služeb. Mnoho těchto zařízení může být spojeno s konkrétním jednotlivcem. To znamená, že shromážděná data by mohla být použita k narušení soukromí osoby, i když každé jednotlivé zařízení bylo navrženo tak, aby bylo zachováno soukromí. Potenciál pro narušení soukromí z neočekávané korelace dat může pocházet z návrhu systému, jelikož je výzvou vyvážit touhu výrobců zařízení IoT a poskytovatelů IoT služeb, aby shromáždili co nejvíce informací za účelem zlepšit své podnikání se zachováním soukromí jednotlivých uživatelů.

Výše popsaná bezpečnostní rizika a výzvy jsou z velké části otázkou koordinace výrobců a poskytovatelů služeb v oblasti IoT, jejichž snaha by měla vyústit v standardizaci a certifikaci bezpečnostních prvků všech IoT zařízení. Tato standardizace by měla zahrnovat jak design, provedení, distribuci, komunikaci i implementaci, tak také životní cyklus a údržbu těchto zařízení z dlouhodobého hlediska.

Kapitola 9

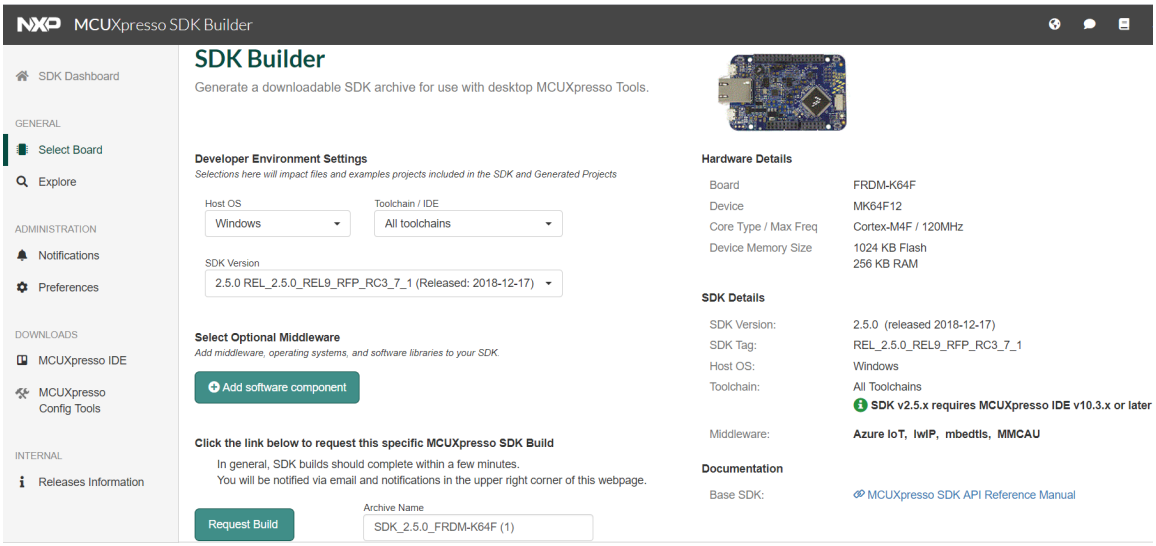
Dosažené výsledky

Výsledky dosažené v této práci byly již úspěšně použity v rámci komerčních řešení některých výrobců, což dokazuje užitnost těchto výsledků.

Produkce NXP Semiconductors

Výsledky byly také použity v rámci SDK (Software Development Kit) výrobce procesorů NXP Semiconductors, díky kterým je realizována podpora Microsoft Azure na některých zařízeních zmíněného výrobce a samotná upravená knihovna Azure IoT je k dostání jako samostatný middleware obsahující demonstrační aplikace realizovány v rámci této práce.

Všechny tyto výstupy práce jsou již v produkci a mohou být k dostání široké veřejnosti skrze online MCUXpresso SDK Builder, jak je vidět na obrázku 9.1.

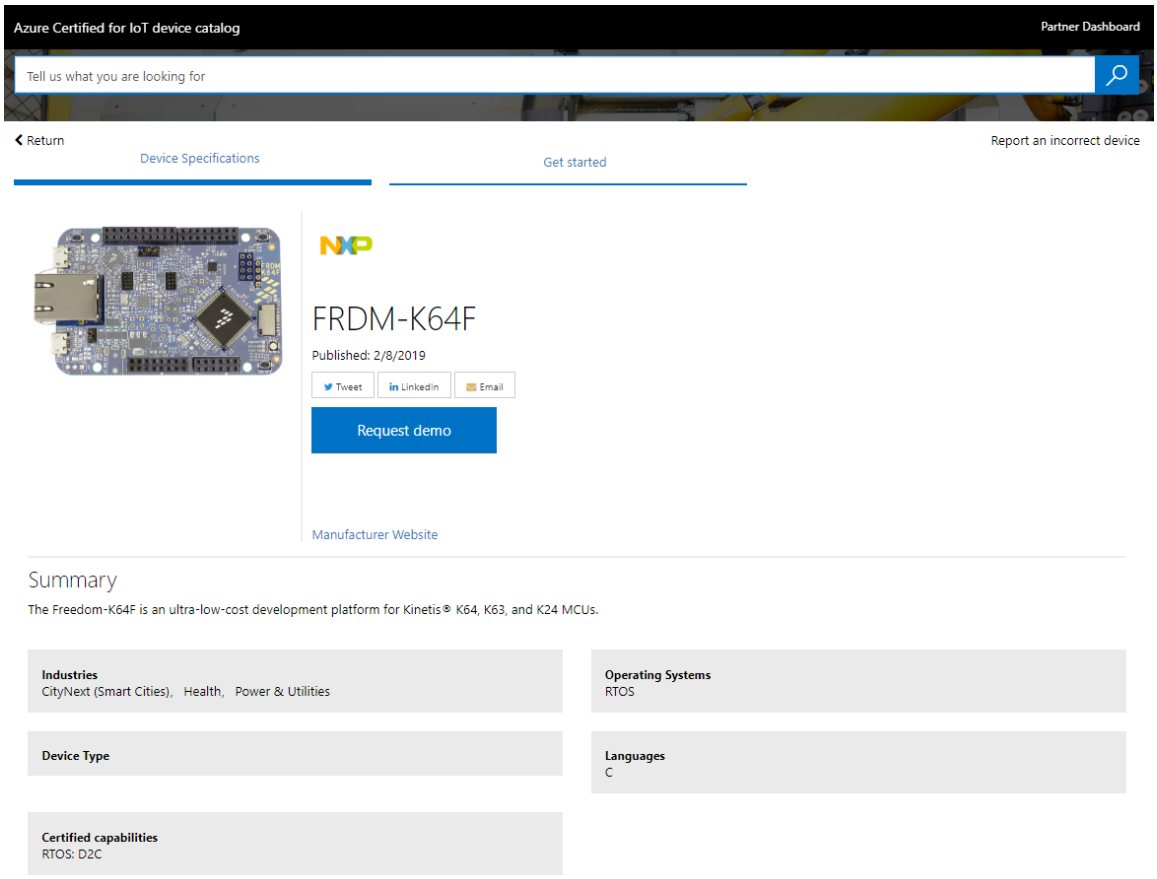


Obrázek 9.1: **MCUXpresso SDK Builder.** Produkční online build generátor s podporou Azure IoT.

Microsoft certifikace

V průběhu vývoje započala také úzká spolupráce se společností Micorsoft, což je samotný poskytovatel cloudové služby. Tato spolupráce kromě výměny implementačních poznatků a doporučení vyústila také v oficiální certifikaci vývojové desky FRDMK64F jako zařízení splňující požadavky stanovené Microsoftem pro použití jejích cloudových IoT služeb. Tento fakt lze ověřit v katalogu podporovaných zařízení na stránkách Microsoftu (viz. obr. 9.2, odkaz zde[1]), kde je k dostání také demonstrační aplikace vytvořená v rámci této práce.

Certifikace dalších vývojových desek různých parametrů a sérií zmíněných v kapitole 7 jsou v procesu schvalování.



Obrázek 9.2: **Katalog podporovaných zařízení MS Azure IoT.**

[1] https://catalog.azureiotsolutions.com/details?title=FRDM_K64F&source=null

Kapitola 10

Závěr

Z výsledků dosažených a prezentovaných v rámci této práce lze dojít k závěru, že výsledné řešení splňuje všechny body zadání. Byla realizována rešerše současných cloudových řešení disponující technologiemi pro realizaci IoT infrastruktury a na základě jejich porovnání byla zvolena platforma Azure od společnosti Microsoft. Na vestavěných zařízeních byla realizována TCP/IP komunikace skrze rozhraní ethernet nad kterou bylo realizováno zabezpečené TLS spojení díky kryptografické knihovně MbedTLS garantující bezpečnou komunikaci. Dále byly dosažené poznatky a zdroje využity pro nasazení samotné knihovny podporující připojení k instanci Azure IoT Hub, což bylo demonstrováno na ukázkové aplikaci. Úspěšné splnění zadání podtrhuje fakt, že tato technologie je již v produkci a doposud byla reálně využita některými zákazníky pro jejich vlastní řešení v rámci chytré domácnosti a IoT.

Dalším milníkem, který je potřeba překonat na cestě vývoje IoT, je bezesporu celkové zabezpečení, ať jíž na úrovni zařízení nebo celkové infrastruktury.

Literatura

[1] Adam Dunkels, L. W.: *LwIP INTRODUCTION*. [Online; navštíveno 20.04.2019].
URL http://www.nongnu.org/lwip/2_1_x/index.html

[2] Amazon: *AWS IoT Core pricing*. [Online; navštíveno 19.04.2019].
URL https://aws.amazon.com/iot-core/pricing/

[3] Amazon: *Getting Started with AWS IoT*. [Online; navštíveno 16.04.2019].
URL https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html

[4] Amazon: *Using Alexa Skills Kit and AWS IoT to Voice Control Connected Devices*. [Online; navštíveno 16.04.2019].
URL https://developer.amazon.com/blogs/post/Tx3828JHC7O9GZ9/Using-Alexa-Skills-Kit-and-AWS-IoT-to-Voice-Control-Connected-Devices

[5] Arm: *ARM Drives Momentum in Microcontrollers with Keil Acquisition.* [Online; navštíveno 29.04.2019].
URL http://www.keil.com/pr/article/1085.htm

[6] Arm: *FRDM-K64F*. [Online; navštíveno 29.04.2019].
URL https://os.mbed.com/platforms/FRDM-K64F/

[7] Arm: *Software development tool suite for high-efficiency microcontrollers.* [Online; navštíveno 29.04.2019].
URL https://developer.arm.com/tools-and-software/embedded/keil-mdk

[8] ARM Limited: *mbedTLS*. [Online; navštíveno 24.04.2019].
URL https://tls.mbed.org/

[9] Dunkels, A.: *lwIP - A Lightweight TCP/IP stack - Summary*. [Online; navštíveno 20.04.2019].
URL https://savannah.nongnu.org/projects/lwip/

[10] Gaddam, K.: *Applying Real-Time Analytics on IoT Data - Azure IoT Hub*. [Online; navštíveno 19.04.2019].
URL https://www.hackster.io/Kishore10211/applying-real-time-analytics-on-iot-data-azure-iot-hub-d5f904

[11] Google: *Cloud IoT Core overview*. [Online; navštíveno 19.04.2019].
URL https://cloud.google.com/iot/docs/concepts/overview

[12] Google: *Cloud IoT Device SDK for Connectivity to IoT Core*. [Online; navštíveno 19.04.2019].
URL https://github.com/GoogleCloudPlatform/iot-device-sdk-embedded-c

[13] Google: *Google Cloud IoT Device SDK for Embedded C Porting Guide.* [Online; navštíveno 19.04.2019].
URL https://github.com/GoogleCloudPlatform/iot-device-sdk-embedded-c/blob/master/doc/porting_guide.md

[14] Google: *Pricing.* [Online; navštíveno 19.04.2019].
URL https://cloud.google.com/iot/pricing

[15] Graz University of Technology: *Vulnerabilities in modern computers leak passwords and sensitive data.* [Online; navštíveno 2.05.2019].
URL https://meltdownattack.com/

[16] IAR Systems: *IAR Embedded Workbench.* [Online; navštíveno 29.04.2019].
URL https://www.iar.com/iar-embedded-workbench/

[17] IBM: *Client libraries and samples for connecting to IBM Watson IoT using Embedded C.* [Online; navštíveno 19.04.2019].
URL https://github.com/ibm-watson-iot/iot-embeddedc

[18] IBM: *IBM Pricing Plans.* [Online; navštíveno 19.04.2019].
URL https://cloud.ibm.com/catalog/services/internet-of-things-platform

[19] Microsoft: *Ceny za Azure IoT Hub.* [Online; navštíveno 19.04.2019].
URL https://azure.microsoft.com/cs-cz/pricing/details/iot-hub/

[20] Microsoft: *How to Port the Azure IoT C SDK to Other Platforms.* [Online; navštíveno 22.04.2019].
URL https://github.com/Azure/azure-c-shared-utility/blob/master/devdoc/porting_guide.md

[21] NXP Semiconductors: *From the INTERNET of THINGS to the INTERNET of TRUST.* [Online; navštíveno 30.04.2019].
URL
https://www.nxp.com/docs/en/white-paper/NXP-FROM-IOT-TO-IOTRUST-WP.pdf

[22] Seaborn, M.: *Exploiting the DRAM rowhammer bug to gain kernel privileges.* [Online; navštíveno 2.05.2019].
URL https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[23] Solcik, V.: *Question: Sending device-to-device messages.* [Online; navštíveno 19.04.2019].
URL https://github.com/Azure/azure-iot-sdk-c/issues/461

Příloha A

Hierarchie knihovny Azure IOT

- **/certs**
  Obsahuje certifikáty potřebné pro komunikaci s Azure IoT Hub.
- **/c-utility**
  - /adapters
    Adaptéry pro účely přenositelnosti knihovny mezi různými platformami.
  - /inc
    Hlavičkové soubory pro zdrojové soubory složky c-utility
  - /pal
    Soubory, jejichž použití je specifické pro operační systém nebo jinou komponentu.
  - /src
    Zdrojové soubory
- **/deps**
  Obsahuje soubory nutné pro běh knihovny, které nejsou její součástí (v tomto případě pouze knihovna Parson pro účel JSON serializace napsaná v jazyce C)
- **/iothub_client**
  Implementace klientu IOT HUB
  - /inc
    Hlavičkové soubory pro zdrojové soubory složky iothub_client
  - /src
    Zdrojové soubory
- **/umqtt** Implementace pro protokol MQTT
  - /inc
    Hlavičkové soubory pro zdrojové soubory složky umqtt
  - /src
    Zdrojové soubory

Příloha B

Obsah CD

- README

  Soubor obsahující popis a základní informace o obsahu disku CD.

- Solcik_Vit_BP.pdf

  PDF dokument bakalářské práce.

- \doc

  Zdrojový kód zprávy (LaTeX).

- \bin

  Složka obsahující výsledné binární soubory aplikací (pozn.: binární kódy jsou zkompilovány s tajným řetězcem připojení, který není ze zjevných důvodů součástí odevzdaných zdrojových kódů).

- \SDK_Azure_IoT

  Repozitář zdrových kódů aplikací, firmware a knihoven.

  - boards\frdmk64f\azure_examples

    Složka obsadující projekty aplikací různých vývojových prostředí a zdrojové soubory aplikací specifické pro konkrétní vývojovou desku. Každá z aplikací má ve složce vlastní README soubor obsahující konfiguraci desky a očekávaný referenční výstup aplikace.

    * \azure_http
      Aplikace demonstrující připojení k Azure IoT Hub pomocí protokolu HTTP.
    * \azure_mqtt
      Aplikace demonstrující připojení k Azure IoT Hub pomocí protokolu MQTT.
    * \service-client-sample
      Java desktop aplikace demonstrující posílaní zpráv na vestavěné zařízení.
    * \azure_http_remote_control
      Aplikace demonstrující připojení a zpracování příhozích zpráv z Azure IoT Hub pomocí protokolu HTTP.
    * \azure_mqtt_remote_control
      Aplikace demonstrující připojení a zpracování příhozích zpráv z Azure IoT Hub pomocí protokolu MQTT.

    - startup_guide.txt
      Soubor popisující kroky pro nastavení Azure IoT Hub.
  - \middleware
    Složka obsahující middleware. Především upravenou knihovnu Azure IoT, MbedTLS a lwIP.
  - \devices
    Složky obsahující specifické soubory pro konkrétní vývojovou desku.
    - \drivers
      Složka obsahující zdrojové soubory ovladačů.

- log_wireshark

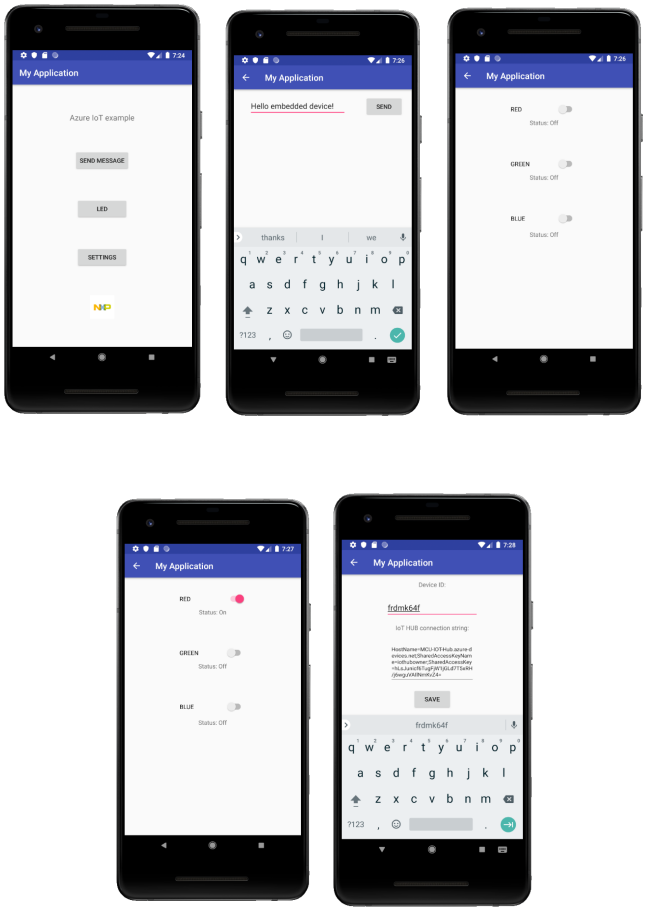
  Soubor pcap zachycených paketů síťové komunikace pomocí SW Wireshark.

Příloha C

Android aplikace

V rámci vývoje servisní aplikace byla vyvinuta snaha vytvořit servisní aplikaci i na platformu Android. Během vývoje byla nalezena chybná implementace komunikace AMQPS v servisní knihovně Azure při užití externí knihovny Proton-J. O této skutečnosti byl Microsoft informován a po téměř roce byl problém vyřešen. Nicméně po další analýze využitelnosti byl další vývoj aplikace přerušen.

Ukázky servisní aplikace jsou vidět na následujících obrázcích C.1:



Obrázek C.1: **Servisní Android aplikace MS Azure IoT.**