

Sem vložte zadání Vaší práce.





**FAKULTA  
INFORMAČNÍCH  
TECHNologiÍ  
ČVUT V PRAZE**

Diplomová práce

## **Detekce anomálií v provozu IoT sítí**

*Bc. Dominik Soukup*

Katedra počítačových systémů

Vedoucí práce: Tomáš Čejka

8. dubna 2018



---

## Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstráňte tento příkaz.



---

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. dubna 2018

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2018 Dominik Soukup. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.*

### **Odkaz na tuto práci**

Soukup, Dominik. *Detekce anomálií v provozu IoT sítí*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.



---

## Abstrakt

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

**Klíčová slova** Nahradte seznamem klíčových slov v češtině oddělených čárkou.

---

## Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

**Keywords** Nahradte seznamem klíčových slov v angličtině oddělených čárkou.



---

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Cíl práce</b>	<b>3</b>
<b>2 Analýza</b>	<b>5</b>
2.1 Architektura IoT sítí . . . . .	5
2.2 Používané komunikační protokoly . . . . .	10
2.3 Bezpečnostní slabiny . . . . .	18
2.4 Možnosti detekce . . . . .	19
2.5 Existující řešení . . . . .	21
2.6 Analýza požadavků . . . . .	22
2.7 Zvolené řešení . . . . .	24
<b>3 Návrh</b>	<b>25</b>
3.1 Architektura . . . . .	25
3.2 Kolektor . . . . .	27
3.3 Detektor . . . . .	28
3.4 Multiplexor a demultiplexor . . . . .	34
3.5 Scénáře útoků . . . . .	35
<b>4 Realizace</b>	<b>37</b>
4.1 Integrace kolektoru . . . . .	37
4.2 Mux a Demux . . . . .	38
4.3 Zpracování zadaných parametrů . . . . .	38
4.4 Výpočet profilu . . . . .	39
4.5 Detekční funkce . . . . .	39
<b>5 Testování</b>	<b>41</b>
5.1 Testovací prostředí . . . . .	41
5.2 Způsoby nasazení . . . . .	41

5.3	Detekce scénářů útoků . . . . .	43
5.4	Test exportu dat . . . . .	45
5.5	Měření položek profilu . . . . .	45
<b>Závěr</b>		<b>47</b>
<b>Literatura</b>		<b>49</b>
<b>A Seznam použitých zkratk</b>		<b>53</b>
<b>B Instalační příručka</b>		<b>55</b>
<b>C Obsah přiloženého CD</b>		<b>57</b>

---

## Seznam obrázků

2.1	Porovnání klasické a fog architektury . . . . .	6
2.2	MQTT architektura . . . . .	11
2.3	CoAP architektura . . . . .	13
3.1	Architektura detekčního systému . . . . .	25
3.2	Návrh kolektoru provozních dat . . . . .	27
3.3	Architektura detektoru . . . . .	28
3.4	Popis konfiguračních parametrů pro jeden záznam . . . . .	31
5.1	Lokální nasazení . . . . .	42
5.2	Oddělené nasazení . . . . .	42



---

# Úvod

Koncept internetu existuje již několik desítek let a pro spoustu lidí se stal nedílnou součástí pracovního i osobního života. V poslední době je možné sledovat stále rostoucí počet zařízení, která jsou do něj zapojena. Tento trend by měl pokračovat i do budoucnosti, a dokonce v ještě větším měřítku. Odhadem je přes 30 miliard připojených zařízení do roku 2020 [1]. Důvodem zrychleného růstu je expanze síťového připojení na veškeré elektronické zařízení a senzory, které umožní vzdálené ovládání a monitorování. Pro označení tohoto trendu se používá termín Internet věcí (Internet of Things, IoT).

Cílem IoT je usnadnit, zlepšit a ušetřit lidskou činnost napříč všemi odvětvími. Uplatnění se nachází zejména ve výrobních podnicích, dopravě nebo běžných domácnostech. Příklad konkrétního nasazení popisuje článek [2], jehož cílem je měření kvality spánku a odhalení případných poruch. Monitorovací systém lze dále propojit například s ovládáním místnosti, které bude reagovat na aktuální úroveň spánku úpravou světla, oken nebo vzduchu.

Dále je upraven model komunikace, který již nevyžaduje zaslání zpráv centrálnímu serveru (north-south), ale podporuje přímou komunikaci mezi připojenými uzly (east-west). Oblast IoT nezahrnuje jen malá a nevýkonná zařízení, ale jeho součástí jsou i výkonná datová centra a pokročilé algoritmy, které vyhodnocují získané informace.

Hlavní hrozbou Internetu věcí je bezpečnost. Dochází zde k přenosu citlivých dat, která slouží k automatizovanému řízení dalších systémů, monitorování prostředí a zabezpečovacím účelům. Zároveň s masivním rozšířením nově připojených zařízení roste riziko vzniku nových útoků a možnosti způsobení větších škod. Příkladem bezpečnostního incidentu je útok na distribuční síť elektrického proudu na Ukrajině, který měl dopad na 225 000 zákazníků [3].

Pro potlačení možného vzniku hrozeb musí být součástí každé dnešní IoT sítě sada procesů, které umožní důvěryhodné získávání validních dat, vzdálenou správu a možnost detekce anomálií jako v běžných IP (Internet Protocol) sítích.





---

## Cíl práce

Cílem magisterské práce je analyzovat množinu aktuálně používaných protokolů pro komunikaci v IoT sítích a identifikovat jejich bezpečnostní zranitelnosti. Při analýze bude věnována pozornost zejména bezdrátovým senzorovým protokolům. Na základě získaných znalostí bude navržen, implementován a otestován algoritmus pro monitorování a automatickou detekci anomálního provozu v IoT sítích. Algoritmus bude možné spustit v prostředí nově vznikající opensource brány BeeeOn, čímž dojde k rozšíření dostupných bezpečnostních funkcí.



# Analýza

Kapitola se zabývá analýzou celkové architektury IoT sítí a způsoby pro její zabezpečení. Postupně je prozkoumán komunikační model, možné bezpečnostní hrozby a existující řešení pro obranu. Na základě analýzy jsou uvedeny funkční a nefunkční požadavky, které jsou kladeny na výsledný program. V závěru jsou vybrány konkrétní technologie pro realizaci.

## 2.1 Architektura IoT sítí

V blízké době se očekává stále větší nárůst zařízení, která jsou připojena k internetu. Dle odhadů by jejich počet měl v roce 2020 překročit 30 miliard [1]. Pro takové množství připojení už není možné, aby každé zařízení komunikovalo přímo se vzdáleným datovým centrem, protože nároky na potřebnou šířku pásma by byly obrovské [4]. Dalším problémem je často velmi omezený výkon připojených prvků, který je nezbytný pro použití bezpečnostních funkcí umožňujících kompletně zabezpečenou komunikaci.

Řešení těchto problémů je do probíhající komunikace přidat několik podvrstev, které umožní přesunout výpočetní výkon blíže ke koncovým zařízením, a tím celý proces zpracování dat provést efektivněji.

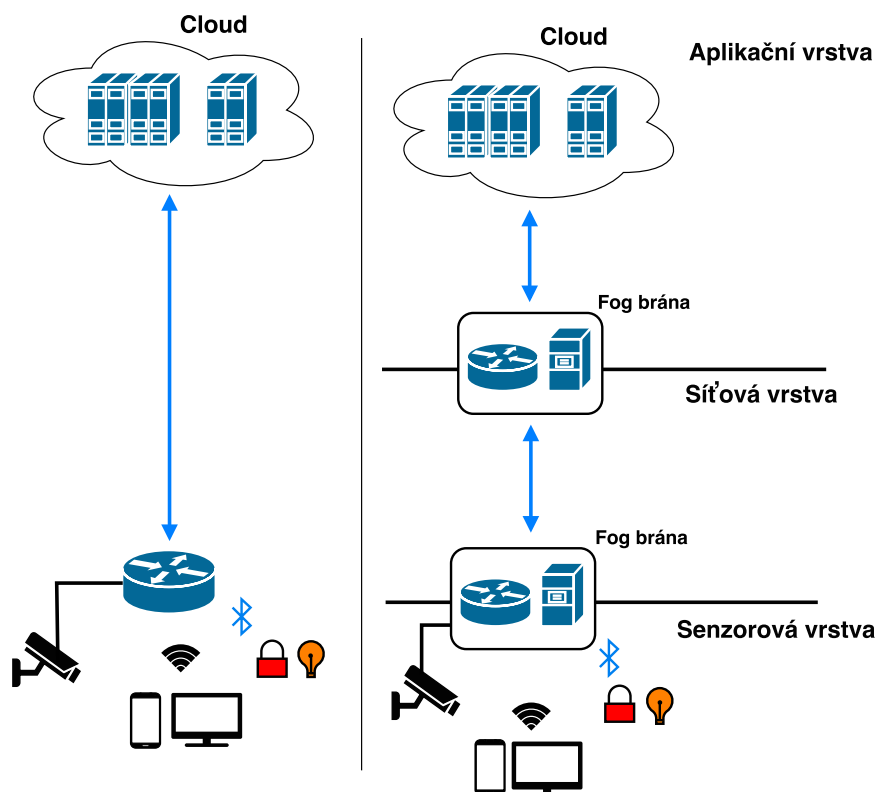
V následujících podkapitolách bude popsán nový model komunikace, který přináší IoT síť.

### 2.1.1 Fog computing

Fog computing je rozšíření cloud computingu, které spočívá v přesunutí výpočetního výkonu blíže k okraji sítě. Rozšíření je umožněno pomocí přidání síťových zařízení, které kromě běžných funkcionalit poskytují i výpočetní výkon pro běh externích programů. Programy je často možné nasadit pomocí kontejnerů nebo samostatných virtuálních strojů, což velmi usnadňuje jejich distribuci [4].

## 2. ANALÝZA

Porovnání klasické a fog architektury z hlediska zpracování dat se nachází na obrázku 2.1. V reálném nasazení může být použito i více fog vrstev, kde každá provádí určitý stupeň předzpracování a řízení dat. Zavedením principů



Obrázek 2.1: Porovnání klasické a fog architektury

fog computingu vznikají pro síť následující výhody:

- **Zlepšení bezpečnosti**

Síťové prvky jsou trvale napájené a připojené k internetu. Podporují pokročilé bezpečnostní funkce, a proto je možné například vytvářet šifrované tunelové spojení pro bezpečný přenos dat [4].

- **Nižší nároky na šířku pásma a latenci**

Odeslaná data z koncových zařízení jsou zpracovávána a filtrována na okraji sítě. Tím je možné rychleji reagovat na přijaté zprávy a snížit nároky na latenci a šířku pásma. Zároveň krátkodobá data mohou být uložena ve fog vrstvě a centrální datové centrum může být využito pro dlouhodobé údaje, které se zpracovávají pokročilými algoritmy pro analýzu dat [4].

- **Jednotná správa**

Při správě sítě už se nemusí přistupovat přímo na koncové prvky, které často komunikují různými protokoly, ale stačí pouze řídit síťová zařízení v jednotlivých fog vrstvách, které odstiňují různorodost protokolů a nabízí standardizovaný přístup. Díky této abstrakci je zároveň zjednodušeno zpracování získaných dat a je umožněno přímé zasílání zpráv mezi koncovými prvky, které používají odlišné komunikační protokoly [4].

Další výhodou je, že výpočetní i síťové funkce jsou zajištěny jedním hardwarovým zařízením, což značně usnadňuje požadavky pro nasazení.

Nevýhodou naopak může být distribuovaná topologie, která má větší nároky na údržbu.

### 2.1.2 IoT brána

IoT brána je síťové zařízení, které je umístěno velmi blízko koncových zařízení a představuje vstup do fog vrstvy. Jejím hlavním cílem je získávat data z připojených zařízení a poskytovat je vyšším vrstvám. Pokud je brána reprezentována výkonnějším síťovým prvkem, tak v něm zároveň může probíhat i základní zpracování dat.

Pro IoT síť je typické, že obsahují velké množství koncových prvků komunikujících různorodými způsoby. Zejména senzory používají protokoly, které nepodporují IP spojení. Důvodem použití této komunikace je často velký důraz na nízkou spotřebu a specifické požadavky na způsob zasílání zpráv. Příkladem protokolů pro senzorové sítě je například: Z-Wave, Bluetooth a Zigbee. Jejich detailní popis se nachází v kapitole 2.2 Používané komunikační protokoly. Tato různorodost vyžaduje, aby brána obsahovala dodatečná rozhraní, které umožní připojení nejrůznějších bezdrátových i drátových koncových prvků.

V současné době existuje mnoho různých bran jejichž parametry se liší dle způsobu nasazení a provozních nároků. Velkým problémem v této oblasti je malý důraz na bezpečnostní funkce, které umožní vzdálené řízení brány, kontrolu provozu a aktualizace programového vybavení. Z těchto důvodů vznikl opensource projekt BeeeOn [5] jehož cílem je vytvořit softwarovou IoT bránu, kterou bude možné spustit na různých hardwarových platformách. BeeeOn brána je navržena modulárně tak, aby byla schopna zpracovávat více rozdílných senzorových protokolů, a tím bude možné provozovat jedno univerzální zařízení namísto několika proprietárních. Zároveň je kladen důraz na bezpečnost, a proto veškeré údaje, které je možné získat o provozu, jsou poskytovány pro analýzu. Nad těmito údaji je postaven detekční algoritmus, který je výsledkem této diplomové práce.

### 2.1.3 Komunikační model a jeho hrozby

Při použití principů popsaných v předchozích kapitolách lze model komunikace rozdělit do následujících vrstev [6]:

- **Senzorová vrstva**

Senzorová vrstva obsahuje veškerá koncová zařízení, které získávají informace ze svého okolí nebo vykonávají potřebnou službu [7]. Tato zařízení jsou připojena kabelově nebo bezdrátově k IoT bráně. K jedné bráně může být připojeno několik prvků, které komunikují odlišnými způsoby. Princip komunikace a možnosti topologie se odlišují na základě použité technologie.

Velkým nebezpečím této vrstvy jsou zejména bezdrátové protokoly, protože při nepoužití zabezpečení může snadno dojít k odposlouchávání nebo úpravám provozu [6]. Dále se zde mohou vyskytovat zařízení, které jsou označeny jako zabezpečené, ale díky starší verzi komunikačního protokolu používají zastaralé bezpečnostní funkce nebo obsahují implementační chyby. Tento případ je velmi nebezpečný, protože vyvolává falešný pocit bezpečí. Útoky se také mohou zaměřovat na prvky s bateriovými zdroji, které mohou být nadměrnou komunikací účelově vybity.

- **Síťová vrstva**

Po zpracování senzorových dat na bráně je nutné získané informace odeslat dalším službám. K tomuto účelu slouží síťová vrstva. Cílem této vrstvy je také umožnit vzdálenou správu brány [7]. Pro výběr konkrétního protokolu je nutné znát rozhraní aplikační vrstvy. Ve většině případů je spojení vytvořeno pomocí protokolu HTTPS (Hypertext Transfer Protocol Secure) nebo technologie VPN (Virtual Private Network). Nad tímto spojením je postavena další služba pro výměnu zpráv. Příkladem může být: MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol) nebo AMQP (Advanced Message Queuing Protocol). Tyto protokoly jsou podrobněji popsány v kapitole 2.2 Používané komunikační protokoly.

Bezpečnostní hrozby síťové vrstvy jsou stejné jako v klasických sítích. Je potřeba dodržet principy důvěry, integrity a dostupnosti. Tímto přístupem je možné předejít útokům jako: DDoS (Distributed Denial Of Service), MITM (Man In The Middle) a podvržení informací. Zároveň je nutné nezapomenout, že se zde většinou vyskytuje M2M (Machine To Machine) komunikace a je důležité použít vhodná komunikační rozhraní [6]. Častým případem bývá zastaralá verze firmware, jehož napadení může vést k nestandardnímu chování nebo dokonce ke kompletnímu převzetí kontroly.

- **Aplikační vrstva**

Aplikační vrstva se stará o ukládání dlouhodobých dat a jejich finální zpracování. Zároveň zobrazuje uživateli zpracované informace a umožňuje provádět konfiguraci celé sítě. Z důvodu její možné rozsáhlosti je kritické, aby správa topologie podporovala automatizaci. [7].

Tato vrstva je umístěna většinou v datovém centru a umožňuje vzdálený přístup. Její bezpečnostní problémy lze přirovnat k problémům cloud computingu. Dle množství požadovaných funkcí může být různě složitá a s rostoucí složitostí se také liší nároky na úroveň zabezpečení. Příkladem možných útoků může být: Buffer Overflow, SQL Injection nebo DDoS.

Mezi obecné problémy patří způsob nasazení. Koncové IoT prvky mají odlišné parametry než běžná uživatelské zařízení. Zároveň se liší i průběh komunikace, která je v IoT méně heterogenní a často založena na M2M. V případě umístění všech koncových prvků do jednoho společného segmentu se komplikuje nastavení bezpečnostních pravidel a také roste možný dopad provedeného útoku. Při úspěšném napadení jednoho prvku útočník získá možnost rozšíření na další uzly ve stejném segmentu.

### 2.2 Používané komunikační protokoly

Jedním z hlavních cílů IoT je možnost vzájemného propojení různých komunikačních protokolů, které umožní automatizovanou výměnu zpráv mezi všemi dostupnými zařízeními. Tímto způsobem je následně možné zefektivňovat a usnadňovat lidskou práci.

V následujících podkapitolách bude popsána množina aktuálně často používaných protokolů včetně jejich bezpečnostních funkcí.

#### 2.2.1 MQTT

MQTT je otevřený síťový komunikační protokol typu publish/subscribe, který byl navržen v roce 1999. Již od návrhu byl zaměřen na nízkou náročnost komunikace a jednoduchost implementace. Díky těmto vlastnostem je velmi vhodný pro IoT a M2M systémy [8].

##### 2.2.1.1 Způsob komunikace

Protokol MQTT je postaven nad transportní vrstvou TCP (Transmission Control Protocol) a využívá model publish/subscribe, který vychází z tradičního způsobu zasílání zpráv typu klient-server. Roli serveru zde plní speciální uzel, který se nazývá broker. Broker je známý všem ostatním klientům, kteří mohou zasílat zprávy pomocí operace *publish* nebo se přihlásit o příjem zpráv díky operaci *subscribe*. Na základě provedených operací broker přijímá zprávy a rozhoduje o jejich přeposlání. Způsob odeslání zprávy závisí na obsažených metadatech.

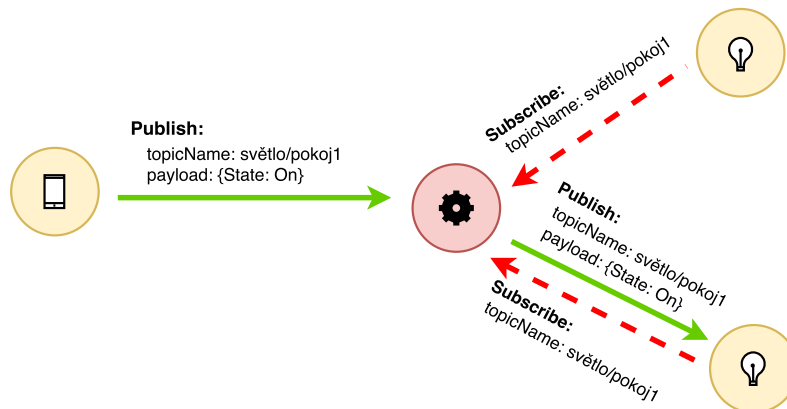
Nejčastěji o směru odeslání rozhoduje předmět (topic) zprávy. Předmět je tvořen jednoduchým UTF-8 řetězcem s hierarchickou strukturou, ve které jsou jednotlivé vrstvy odděleny dopředným lomítkem a každá z nich musí obsahovat minimálně jeden znak (např. domov/přízemí/světlošatna). V předmětu zprávy mohou být některé vrstvy nahrazeny zástupnými symboly `+` a `#`. Symbol `+` dokáže nahradit pouze jednu úroveň předmětu libovolným řetězcem a symbol `#` umožňuje zastoupit více úrovní. Díky těmto symbolům mohou klienti odeslat nebo přijímat zprávy z více témat. Schéma topologie a ukázka využití předmětů zpráv se nachází na obrázku 2.2.

Další užitečnou položkou protokolu MQTT je QoS (Quality of Service), která může nabývat hodnot 0, 1 nebo 2.

- **QoS 0**

Veškeré zprávy jsou odesílány bez potvrzení a žádným způsobem není zvýšena úroveň spolehlivosti, která je shodná se spolehlivostí protokolu TCP.





Obrázek 2.2: MQTT architektura

- **QoS 1**

Pomocí potvrzování zajišťuje, že každá zpráva bude příjemci doručena alespoň jednou.

- **QoS 2**

Umožňuje, aby každá zpráva byla spolehlivě doručena právě jednou.

Hodnota QoS se nastavuje vždy mezi dvěma uzly při navazování spojení. Z pohledu brokeru se může stát, že přijatá a odeslaná zpráva mají jiné QoS. Úrovně 1 a 2 dále umožňují perzistentní ukládání zpráv v případě, že příjemce je nedostupný. Zároveň platí, že s vyšší úrovní roste i režie komunikace [9].

### 2.2.1.2 Bezpečnost

Zabezpečení protokolu MQTT je možné rozdělit do následujících vrstev:

- **Síťová vrstva**

Veškerá komunikace je postavena nad TCP/IP, a proto lze probíhající komunikaci zapouzdřit pomocí VPN připojení jako v běžných počítačových sítích. Kvůli větším nárokům na výkon je toto řešení vhodnější pro výkonnější zařízení jako jsou například IoT brány, které mohou s brokerem navázat site-to-site spojení [10].

- **Transportní vrstva**

Na této úrovni se využívá šifrování provozu pomocí protokolu TLS (Transport Layer Security). Omezením této metody jsou požadavky na výkon, které mohou být poměrně vysoké pokud nastává časté navazování spojení [10].

- **Aplikační vrstva**

Samotný protokol MQTT nedefinuje žádné šifrovací mechanismy na aplikační úrovni. Zabezpečení dat zde musí zajistit uživatel ještě před jejich zapouzdřením do MQTT zprávy. Ovšem tímto způsobem je možné šifrovat jen tělo zprávy a hlavička zůstává nezměněná.

Pro autentizaci je možné využít ověření pomocí jména a hesla nebo x.509 certifikátu. Jméno a heslo je přenášeno nešifrovaně, a proto je vhodné tuto metodu doplnit se zabezpečením síťové nebo transportní vrstvy. Autentizaci pomocí certifikátů je možné využít jen v případě použití TLS. Tato metoda je vhodnější pokud všechna zařízení jsou po jednotnou správou a je možné automatizovat distribuci klientských certifikátů.

Dále je na straně brokeru možné definovat pravidla pro autorizaci. Tyto pravidla přiřazují klientů oprávnění pro provedení operací publish a subscribe nad příslušnými tématy [10].

### 2.2.2 CoAP

CoAP je otevřený přenosový protokol určený pro komunikaci síťových zařízení s velmi omezeným výkonem. Návrh vychází z RESTful (Representational State Transfer) principů, tudíž jeho použití je velmi vhodné v prostředích s již existujícím webových rozhraním, do kterého se snadno integruje. [11]

#### 2.2.2.1 Způsob komunikace

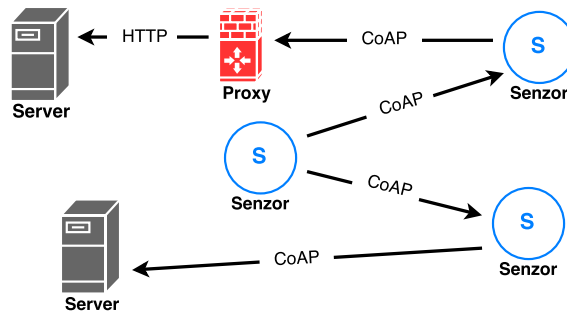
Komunikace je postavena nad protokolem UDP (User Datagram Protocol) a vychází z modelu request/response, který se využívá u protokolu HTTP (Hypertext Transfer Protocol). Oproti protokolu HTTP je zde omezen počet možných operací a komunikace probíhá asynchronně. Dle důležitosti odesílané zprávy je možné určit, zda se má zasílat potvrzení či nikoli. Protože veškerá komunikace probíhá nad protokolem UDP, tak přenášené zprávy obsahují položku Message ID, která dokáže ošetřit duplikaci přijatých dat.

Pro zajištění integrace s běžnými webovými službami a zachování nízkých nároků se v CoAP topologii velmi často vyskytují proxy servery. Tyto servery mohou plnit funkci běžných reverzních a dopředných proxy, mapování mezi CoAP a HTTP protokolem nebo vyvažování zátěže. Způsob komunikace a schéma možné způsoby propojení jsou na obrázku 2.3. Senzor reprezentuje koncový prvek s omezeným výkonem.

Ve specifikaci protokolu CoAP jsou definovány následující operace:

- **GET**

Metoda *GET* vrací aktuální stav požadovaného zdroje, který je identifikován pomocí URI (Uniform resource identifier). Tato operace je vždy bezpečná a idempotentní.



Obrázek 2.3: CoAP architektura

- **POST**

*POST* požadavek obsahuje ve svém těle novou reprezentaci cílového zdroje a požaduje jeho zpracování. Funkce, která novou reprezentaci přijímá je definovaná na cílovém uzlu s příslušným URI. Výsledkem je vytvoření nového zdroje nebo aktualizace původního. Tato metoda není z pohledu zpracování bezpečná ani idempotentní.

- **PUT**

Tato metoda specifikuje nový stav cílového zdroje, který specifikován použitým URI. Zdroj se buď vytvoří, nebo v případě jeho existence aktualizuje. Provedení operace není bezpečné, ale je idempotentní.

- **DELETE**

Operace *DELETE* požaduje smazání zdroje s příslušným URI. Tato metoda není bezpečná, ale je idempotentní.

Vyhledání potřebného zdroje je v topologii protokolu CoAP možné provést pomocí znalosti jeho URI nebo odesláním multicastového požadavku na definovanou skupinu uzlů. Možnost vyhledávání cílových zdrojů je velmi důležitá zejména v M2M prostředích [11].

### 2.2.2.2 Bezpečnost

Samotný protokol nijak nedefinuje možnost autentizace a autorizace. V případě potřeby je nutné tyto mechanismy implementovat v aplikačním kódu.

Pro zajištění šifrování provozu nabízí CoAP následující režimy:

- **NoSec**

Tento mód neobsahuje žádnou úroveň zabezpečení a veškeré zprávy jsou zasílány v otevřené podobě.

- **PreSharedKey**

V tomto případě se naváže spojení pomocí protokolu DTLS (Datagram Transport Layer Security), které využívá symetrické šifrování. Předsdílený klíč musí být známý všem uzlům před zahájením komunikace.

- **RawPublicKey**

Tento režim také navazuje zabezpečené spojení pomocí protokolu DTLS, ale místo symetrické šifry se používá asymetrická.

- **Certificate**

Mód *Certificate* rozšiřuje *RawPublicKey* o přidání certifikátu.

Podobně jako u protokolu MQTT je i zde možné ochránit provoz na síťové vrstvě pomocí VPN. Při nasazení libovolného režimu zabezpečení je nutné počítat s většími nároky na výkon, které musí klient splňovat pro navazování a udržování spojení [11].

### 2.2.3 Z-Wave

Z-Wave je bezdrátový komunikační protokol určený pro senzorové sítě, který vysílá v subgigahercových pásmech ISM (Industrial, Scientific and Medical). Veškeré komunikační prvky jsou certifikovány aliancí Z-Wave, která zároveň poskytuje technickou dokumentaci a licence pro vývoj. V současné době existují certifikace Z-Wave a Z-Wave Plus. Z-Wave Plus zařízení obsahují nový chipset, který vylepšuje komunikační parametry sítě a zároveň je zpětně kompatibilní se staršími modely. [12] Otevřená implementace celého protokolu se nazývá OpenZWave [13], která je vyvíjena komunitou [14].

#### 2.2.3.1 Způsob komunikace

V Z-Wave síti se může maximálně vyskytovat 232 uzlů, mezi kterými se vždy nachází jeden označený jako kontroler. Pro přidání libovolného zařízení do sítě musí být nejprve provedeno přímé párování s kontrolerem. Během párovacího procesu nový prvek získá vlastní 8 bitový identifikátor (*Node ID*) a unikátní 32 bitový identifikátor sítě (*Home ID*), který má již od výroby kontroler uložen v nepřepisovatelné paměti. Následné zasílání zpráv už nemusí probíhat přímo mezi senzorem a kontrolerem, ale zprávy mohou být přeposílány sousedními prvky, které jsou trvale napájené, čímž se velmi zvyšuje možná rozloha sítě. Pro odebrání libovolného zařízení je opět nutné zajistit přímé spojení s kontrolerem a spustit odstraňovací proces [14].

V rámci specifikace Z-Wave [15] je definován mechanismus pro určení dostupných příkazů. Každý senzor obsahuje svou definici tříd funkcionalit (*Command Class*), které obsahují dostupné příkazy a formát odpovědi. Tyto definice jsou předány kontroleru během procesu párování. Příkladem může být třída *Binary Switch*, která obsahuje příkazy:

- *SET* - odesílá kontroler pro nastavení hodnoty
- *GET* - odesílá kontroler pro získání hodnoty
- *REPORT* - odesílá senzor jako odpověď na dotaz *GET*

Při posílání zpráv je vždy vyžadováno potvrzení. V případě neobdržení potvrzení se vysílání opakuje. Po třetím neúspěšném pokusu je požadavek zahozen.

### 2.2.3.2 Bezpečnost

Původní verze protokolu Z-Wave umožňovala volitelně využívat šifrování pomocí 128 bitového AES (Advanced Encryption Standard). Výměna symetrického klíče v této verzi probíhá během počátečního párování, kdy je šifrování pomocí výchozího klíče, který je uložen ve firmwaru. Vzhledem k tomuto postupu je dobré provádět počáteční párování na bezpečném místě, kde nemůže dojít k odposlechu [16].

V roce 2016 Z-Wave aliance vydala nový S2 (Security 2) framework, který vylepšuje bezpečnostní funkcionality a od roku 2017 je jeho použití povinné pro všechny nově certifikovaná zařízení.

S2 framework umožňuje zařízení rozdělit do následujících skupin s rozdílnými šifrovacími klíči:

- **Access Control** - nejdůvěryhodnější třída, která obsahuje bezpečnostní prvky jako jsou například zámky, které zároveň podporují autentizaci
- **Authenticated** - skupina určená pro běžné senzory, které podporují autentizaci
- **Unauthenticated** - třída pro ostatní prvky, které nepodporují autentizaci.

Autentizace probíhá pomocí PIN (Personal Identification Number) nebo QR (Quick Response) kódu. Výměna klíče je založena na algoritmu ECDH (Elliptic-curve Diffie–Hellman), který zajišťuje dostatečnou úroveň bezpečnosti během párovacího procesu [14].

### 2.2.4 BLE (Bluetooth Low Energy)

BLE je bezdrátový protokol určený pro senzorové sítě s důrazem na nízkou spotřebu. Byl představen v roce 2010 jako součást specifikace Bluetooth 4.0 a je nekompatibilní s původními verzemi. Za vývoj a údržbu protokolu je odpovědná skupina Bluetooth SIG (Special Interest Group). V současné době je nejnovější verzí Bluetooth 5 [14].

### 2.2.4.1 Způsob komunikace

BLE vysílá v bezlicenčním ISM pásmu na frekvencích od 2.4 GHz do 2.4835 GHz. Specifikace protokolu vychází z Bluetooth, ale zejména díky změnám parametrů v rádiové vrstvě jsou navzájem nekompatibilní. Do verze 4 umožňuje navázat pro koncová zařízení pouze jedno spojení, a proto vytváří hvězdovou topologii s jedním centrálním prvkem. Od verze 5 je možné využívat více připojení a lze vytvořit flexibilnější mesh topologii [14].

Před zahájením komunikace mezi centrálním prvkem a senzory je nutné nejprve provést párování, které probíhá v následujících krocích:

- **Vysílání žádostí**

Při spuštění párování začne koncový prvek na kanálech určených pro propagaci všesměrově vysílat žádosti o připojení, které obsahují název zařízení, jméno výrobce a podporované služby [17].

- **Přijímání žádostí**

Pokud je centrální prvek přepnutý do párovacího režimu, tak naslouchá příchozím požadavkům, které zobrazuje uživateli. Po výběru správného zařízení se ukončí mód naslouchání a začne se navazovat spojení [17].

- **Inicializace připojení**

Během této fáze se obě strany domlouvají na parametrech komunikace [17].

- **Komunikace**

Po úspěšném navázání spojení je možné na základě dostupných služeb provádět zasílání zpráv [17].

### 2.2.4.2 Bezpečnost

BLE umožňuje volitelně používat šifrování pomocí AES, jehož šifrovací klíč je vytvořen během párování v části inicializace připojení. Velmi důležité je zabezpečit výměnu sdíleného klíče, která až do verze 4.1 není bezpečná, protože neumožňuje ochranu před odposloucháváním. Vylepšení přichází až od verze 4.2, ve které lze využít ECDH [14].

Vzhledem k různorodosti možných typů zařízení definuje BLE následující kategorie určující způsob výměny klíče:

- **Just Works**

Nejjednodušší třída, která provádí výměnu automaticky a zároveň má nejmenší nároky na připojované zařízení, které nemusí podporovat autentizaci. Díky chybějící autentizaci je tato metoda zranitelná vůči MITM útokům [14].

- **Out of Band**

V této kategorii jsou veškeré klíče vyměňovány odlišným komunikačním kanálem např. přes NFC (Near Field Communication). Celková bezpečnost této metody závisí na důvěryhodnosti použitého kanálu.

- **Passkey**

Tato metoda vylepšuje *Just Works* o autentizaci, která spočívá v uživatelském zadání šestimístního kódu. Pro ochranu před odposlouchávání musí být použit algoritmus ECDH, který je dostupný až od verze 4.2 [14].

- **Numeric Comparison**

Využití tohoto způsobu párování je možné pouze od verze 4.2. Dochází zde k rozšíření metody *Just Works* o jeden kontrolní krok, který slouží jako ochrana před MITM útokem. Po výměně klíčů každé zařízení vygeneruje šestimístný číselný kód, který následně zobrazí uživateli a čeká na jeho potvrzení [14].

### 2.3 Bezpečnostní slabiny

Na základě provedené analýzy aktuálně používaných protokolů 2.2 budou v této kapitole popsány jejich bezpečnostní slabiny, které budou později využity při návrhu detekčních algoritmů. Jelikož je tato práce zaměřena na senzorové protokoly nekomunikující přes IP, budou v podkapitolách popsány slabiny protokolů Z-Wave a BLE.

#### 2.3.1 Z-Wave

Hlavním problémem jsou zařízení certifikovaná před březnem 2017, jelikož nepodporují S2 framework. Tyto prvky nemusí při své komunikaci využívat žádnou formu šifrování a síť se tak stává velmi zranitelnou. V minulosti již bylo provedeno několik útoků, které pomocí Scapy-Radio projektu [18] nebo knihovny OpenZWave byly schopny ovládat jednotlivé senzory.

Při využití šifrování je velmi zranitelná doba během párovacího procesu, jelikož je při výměně šifrovacích klíčů použit výchozí klíč uložený ve firmwaru zařízení. Zároveň se u jednoho typu zámku podařilo objevit implementační chybu [16], která umožňovala vnutit nový šifrovací klíč a převzít tak kontrolu nad senzorem.

Při využití S2 frameworku dosud nebyly objeveny žádné zranitelnosti, ovšem tento framework je poměrně nový a většina aktuálně používaných i nabízených zařízení byla certifikována ještě před jeho zavedením.

#### 2.3.2 BLE

Velkou hrozbou jsou zařízení s verzí 4.1 nebo nižší, protože nepodporují žádnou ochranu před odposloucháváním a MITM útokům v době párování. Od verze 4.2 je již pro výměnu klíčů použit ECDH algoritmus, který zabraňuje možnému odposlouchávání, ale v případě použití párovací metody, které nepodporuje autentizaci není zajištěna ochrana před MITM útoky. [14]

Dalším problémem jsou samotní výrobci, kteří často nevyužívají bezpečnostní funkce protokolu [19] nebo implementují vlastní způsoby zabezpečení na aplikační úrovni. Tímto dochází k nezabezpečení fáze párování a často se vyskytují i implementační chyby, které přinášejí další zranitelnosti [17] a umožňují útočníkovi získat kontrolu nad celým provozem. [14]

Nevýhodou je velmi dlouhá a komplikovaná specifikace protokolu, která vede k implementačním chybám samotného BLE [20]. Tím vzniká nebezpečí, že i u výrobce, který využívá všech bezpečnostních funkcionalit se mohou vyskytovat zranitelnosti díky chybné implementaci komunikačních vrstev protokolu. [14]



## 2.4 Možnosti detekce

V běžných IP sítích se pro detekci hrozeb nejčastěji používají IDS (Intrusion Detection System) a IPS (Intrusion Prevention System) systémy. Tyto služby rozšiřují koncept klasického firewallu, který blokuje nebo povoluje síťový provoz na základě statických pravidel, o podrobnější sledování datových toků, které jsou zablokovány na základě nestandardního chování.

IDS/IPS může být reprezentováno samostatným hardwarovým zařízením nebo softwarovým programem, který může být dále rozšířen o monitorovací sondy, které se starají pouze o sběr dat a jejich odesílání pro následnou analýzu. Zároveň se může lišit i umístění v síti. Detekční systémy lze provozovat před hraničním směrovačem a detekovat tak kompletní příchozí a odchozí data nebo za hraničním směrovačem, což umožní vyhodnocovat jen vyfiltrovaný provoz. Případně je možné nasadit IDS/IPS přímo na koncové stanice, kde kromě síťových dat lze získávat i informace o běhu systému. Poslední způsob poskytuje nejvíce detailní možnost analýzy, protože se provádí na místě vzniku komunikace a případný útok je možné zastavit již při jeho začátku a zabránit tak případnému rozšíření. Nevýhodou takového nasazení je ztráta celkového pohledu na síť. Při zavedení IDS/IPS systému je z těchto důvodů dobré kombinovat způsoby nasazení a umožnit vzájemné sdílení detekovaných incidentů.

Při zaměření na zpracovávání datových toků lze detekční systémy rozdělit do dvou základních kategorií:

- **Detekce anomálií**

Tato metoda je založena na statistickém modelování. Nejprve se vytvoří profil běžného chování sítě, který se následně porovnává s aktuálním provozem. Dle použité metody se může profil běžného provozu průběžně aktualizovat. Pokud měřené údaje síťového provozu překročí hodnotu stanoveného profilu nad definovaný limit, tak je detekován incident. Výhodou metody anomálií je její uplatnění i na dosud neznámé útoky. Tento princip zároveň způsobuje větší míru falešných poplachů, a proto je nutné jejich pečlivější ověření. Příkladem detekčních metod je: strojové učení, časové řady nebo stavové automaty [21].

- **Detekce signatur**

V případě použití této metody je využíváno signatur (profilů) předem známých útoků. Výhodou je, že díky popsáním signaturám je tato metoda poměrně přesná a detekuje málo falešných incidentů. Naopak nevýhodou je, že není možné detekovat nové druhy útoků, pro které není známý profil. Problémem také je, že signatury musí být uloženy na nějakém perzistentním úložišti, které je dostupné lokálně nebo vzdáleně [21].

## 2. ANALÝZA

---

IoT sítě k běžnému IP provozu přidávají senzorové protokoly, jejichž chování je také dobré monitorovat, protože jsou připojeny do počítačové sítě a mohou být zneužity při útocích. Bohužel detekční metody nejsou pro tyto sítě moc rozšířené, a tím dochází ke zvýšení rizika a dopadu možných útoků. Pro určení incidentů lze využít stejných principů jako v IP sítích, ale liší se způsob získávání dat pro analýzu. Pro sběr informací lze využít následující přístupy:

- **Testbed**

Tato metoda spočívá ve vytvoření specializovaného prostředí, ve kterém se nachází pouze testované a měřicí zařízení. Cílem je ověřit, že nově připojovaný senzor splňuje veškeré bezpečnostní požadavky a neobsahuje žádné známe zranitelnosti. Měřicí prvky reprezentují nástroje, které jsou schopné odposlouchávat komunikaci a využívají se také například při automatizovaných penetračních testech.

- **Externí sonda**

Funkce sondy je stejná jako v IP sítích. Jedná se o samostatné zařízení umístěné v síti, které umožňuje sledovat probíhající komunikaci a odesílat získané údaje ke zpracování. Výhodou je velké množství různých dat, které lze získat, ale zároveň komplikací je šifrovaný provoz a často i cena kvalitní sondy. Dalším využitím může být honeypot, kdy se sonda tváří jako zranitelný prvek a reportuje veškeré pokusy o útok.

- **Provozní statistiky**

Posledním způsobem je sběr dat z příslušných rozhraní na IoT bráně. Tento postup nevyžaduje použití žádného dalšího zařízení, ale potřebuje, aby brána umožňovala získávat tyto statistiky. Statistiky nejsou tak podrobné jako u externí sondy, ale výhodou je, že získávání aktuálních dat o provozu je poměrně nenáročné.

Každá z předchozích metod používá jiný styl sběru dat. Při reálném použití je dobré vyhodnotit bezpečnostní rizika a hrozby, podle kterých lze jednotlivé metody vhodně kombinovat.

## 2.5 Existující řešení

V současnosti se veškerá dostupná řešení zaměřují na detekci útoků v IP protokolech. Mezi současnými IDS/IPS existují signatury pouze pro SCADA (Supervisory Control And Data Acquisition) protokoly. Router Turris Omnia umožňuje nasadit systém Suricata, pro který nabízí rozšíření PaKon [22]. Toto rozšíření zpracovává data ze Suricaty, které následně ukládá v přehledné formě. Díky tomu uživatel získá podrobný přehled o stavu provozu. Získaná data může dále využívat například k vylepšení stávajících bezpečnostních pravidel. Nevýhodou tohoto řešení jsou vyšší hardwarové požadavky, a proto ho nelze provozovat na branách s omezenými prostředky. Další komplikací může být centralizovaná architektura a zaměření pouze na IP provoz.

Na základě provedené analýzy nebylo nalezeno žádné řešení, které umožňuje vyhodnocovat provoz aktuálně používaných IoT protokolů s ohledem na možné omezení hardwarových prostředků.

### 2.6 Analýza požadavků

Podstatnou částí návrhu výsledného řešení je přesné určení požadavků, které se dělí na funkční a nefunkční. Funkční požadavky specifikují funkcionality kladené přímo na vznikající program, zatímco nefunkční spíše určují omezení vlastností systému a architekturu návrhu. V následující kapitole budou popsány nároky na vznikající detekční nástroj, které vycházejí z obsahu zadání této práce a provedené analýzy.

#### 2.6.1 Funkční požadavky

Kapitola popisuje funkční požadavky, které jsou od detekčního nástroje očekávány.

- **Sběr informací o provozu**

Program bude umožňovat sběr dat na IoT bráně o aktuálním provozu z dostupných komunikačních rozhraní. Získané informace bude možné pomocí NEMEA [23] (Network Measurements Analysis) frameworku předávat do dalších modulů.

- **Detekce anomálií**

Vytvořený detekční algoritmus bude schopen na základě získaných dat odhalit definované anomálie, které reprezentují neočekávané změny v síti.

- **Konfigurace způsobu detekce**

Detekční modul bude umožňovat nastavení parametrů pro jednotlivé zpracovávané položky pomocí konfiguračního souboru. Tyto parametry budou následně sloužit jako vstup pro vytvořený detektor.

- **Zpracování získaných informací**

Kromě analýzy dat a hlášení nalezených incidentů bude také možné zpracovaná data pravidelně exportovat do dalších rozšiřujících modulů.

#### 2.6.2 Nefunkční požadavky

Kapitola obsahuje nefunkční požadavky, které jsou kladené na výsledný nástroj.

- **Rozšiřitelnost**

Architektura celého řešení bude navržena tak, aby bylo možné rozšíření o nové způsoby detekce anomálií a další typy provozních dat.

- **Flexibilita nasazení**

Návrh způsobu detekce umožní fyzicky oddělit komponentu zajišťující sběr dat a komponentu vyhodnocující provoz. Díky tomu bude možné

provádět analýzu i na IoT branách s velmi omezeným prostředky, protože tyto brány budou sloužit jako sondy, které budou posílat data do externího detektoru s dostatečným výkonem k provedení analýzy.

- **Operační systém**

Výsledné řešení bude implementováno a otestováno na operačním systému Ubuntu 16.04, na kterém bude nasazena IoT brána BeeeOn a framework NEMEA. Zároveň budou při návrhu a implementaci vybírány takové technologie, aby bylo možné vytvořený program spustit pod distribucí OpenWrt, které je velmi rozšířena mezi síťovými prvky.

### 2.7 Zvolené řešení

Obsahem této kapitoly je popis zvoleného řešení, které bylo na základě provedené analýzy určeno pro realizaci výsledného nástroje.

Vytvořený detekční algoritmus bude zaměřen na senzorové IoT protokoly, protože v současné době neexistuje řešení, které by to umožňovalo. Pro analýzu budou použity protokoly Z-Wave a BLE, které jsou v dnešních sítích velmi rozšířené. Algoritmus bude umístěn přímo na IoT bráně, která je ideálním místem, protože se nachází velmi blízko koncových zařízení a má dostatečný výkon k vyhodnocování provozu. Konkrétně bude použita brána BeeeOn, protože v současné době jako jediná umožňuje sběr provozních dat o senzorových protokolech.

Pro účely detekce budou sbírány jen informace dostupné z lokálních rozhraní brány. Je velice pravděpodobné, že z hlediska reálného nasazení bude tato varianta nejčastější, protože nevyžaduje žádná dodatečná monitorovací zařízení. Zároveň bude umožněno rozšíření i pro další způsoby získávání dat.

Vyhodnocování dat bude probíhat metodou detekce anomálií, která je vhodná pro statistickou povahu dat, má menší nároky na množství dostupných prostředků a umožňuje rozpoznat i neznámé útoky. Metoda bude realizovaná pomocí časových řad jejichž parametry bude možné upravit dle potřeby.

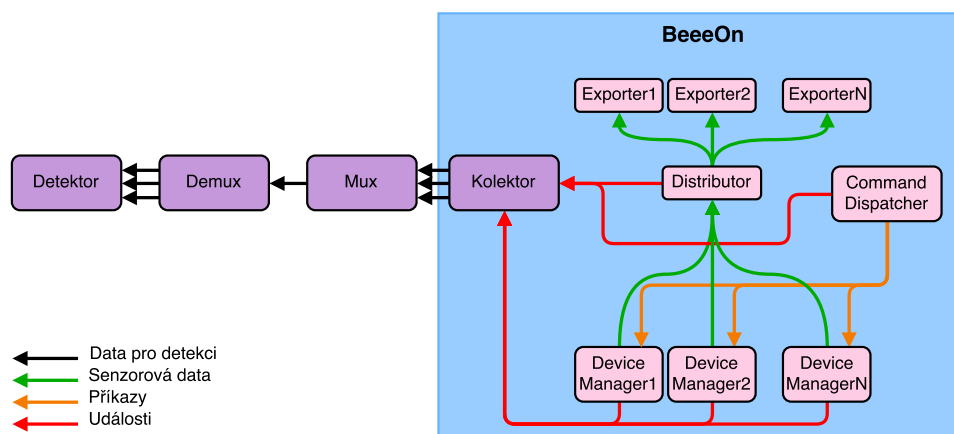
Jako programovací jazyk bude použit C++, protože podporuje objektový přístup a úspornou implementaci na paměť i procesorový čas. Zároveň tento jazyk je použit i v bráně BeeeOn, tudíž bude usnadněna integrace. Pro zajištění flexibility a možnosti dalšího rozšiřování bude využito systému NEMEA, který také podporuje jazyk C++.

## Návrh

Kapitola se zabývá návrhem způsobu detekce anomálií v IoT sítích, který s pomocí NEMEA systému rozšiřuje bránu BeeeOn. Nejprve je popsána celková architektura a možné způsoby nasazení. Dále následuje popis jednotlivých komponent detekčního systému. V závěru jsou identifikovány možné scénáře útoků.

### 3.1 Architektura

Cílem této práce je vytvoření programu, který bude schopen detekovat anomální provoz v IoT sítích. Vzhledem k hardwarovým omezením, které můžou na IoT branách nastat, je architektura navržena tak, aby měla co nejmenší nároky na dostupné prostředky. Schéma nasazení detekčního systému se nachází na obrázku 3.1



Obrázek 3.1: Architektura detekčního systému

### 3. NÁVRH

---

Implementace BeeeOn brány obsahuje pro zpracování sensorových dat následující komponenty:

- **DeviceManager**: komponenta definovaná pro každý sensorový protokol, která implementuje veškerou komunikaci a zpracování dat
- **Distributor**: přijímá data od *DeviceManageru*, která následně předává příslušnému *Exporteru*
- **Exporter**: implementuje protokol, kterým jsou data odesílány z brány
- **CommandDispatcher**: komponenta, která přijímá uživatelské příkazy a distribuuje je cílovým komponentám

Každá z komponent v BeeeOn bráně navíc využívá návrhový vzor *Observer*, pomocí kterého jsou definovány události poskytující informace o každé komponentě. Tímto způsobem bude vytvořený kolektor získávat data o provozu, která převede do formátu UniRec (Unified Record) a pomocí systému NEMEA je odešle k analýze.

Exportované informace o provozu bude přijímat detektor, který následně provede jejich zpracování a vyhodnocení stavu. Všechny vytvořené komponenty vytvořené pro detekci budou používat rozhraní NEMEA. Díky tomu bude možné komponenty flexibilně provozovat na jednom nebo více různých zařízeních. Oddělené nasazení je důležité zejména pro brány s omezenými prostředky, které mohou provádět pouze export dat a o vyhodnocení se bude starat odlišné zařízení s dostatečným výkonem. V případě provozování více bran lze brány používat jako exportéry a veškerá získaná data zpracovávat centrálně.

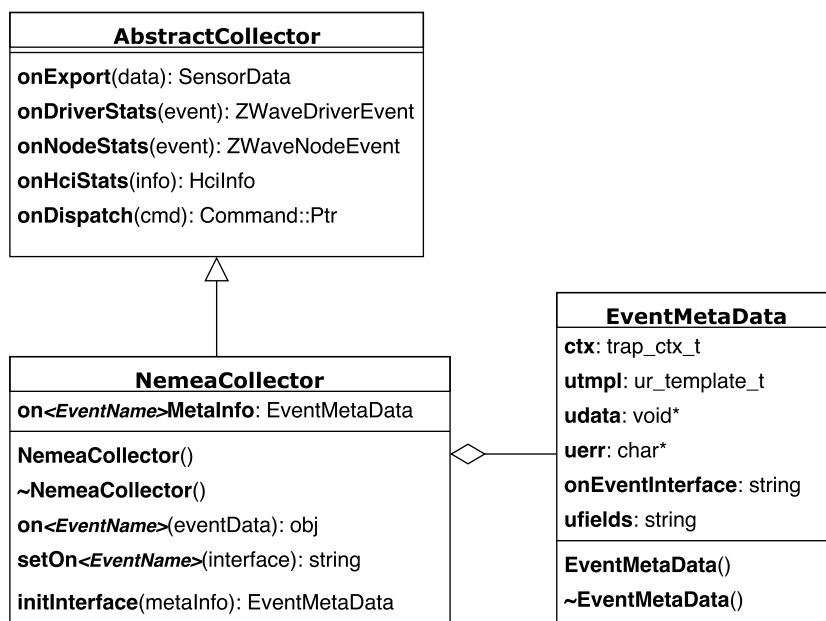
Kolektor pro každou událost vytvoří samostatné výstupní rozhraní. Protože událostí může být velké množství, tak bude vytvořen modul *Mux*, který všechny výstupní rozhraní spojí do jednoho. Pro zpětné rozdělení na jednotlivá rozhraní bude sloužit modul *Demux*. Tyto moduly budou velmi užitečné zejména v případě kdy kolektor a detektor budou na rozdílných síťových prvcích, protože údaje o provozu budou přenášeny přes počítačovou síť a bude vyžadováno zabezpečení všech odchozích rozhraní. S využitím *Mux* a *Demux* modulů bude stačit zabezpečit pouze jedno sjednocené rozhraní.

Výhodou návrhu řešení je flexibilita a modularita. Každá komponenta vždy samostatně pokrývá pouze jednu část detekčního systému a díky NEMEA se každá z nich může nacházet na různých síťových prvcích. Zároveň v případě potřeby lze vytvořit další moduly, které budou rozšiřovat stávající funkcionalitu. Podrobnější popis návrhu nově vytvořených komponent detekčního systému se nachází v následujících kapitolách.



## 3.2 Kolektor

Úkolem kolektoru bude sběr dostupných dat a jejich odeslání k následné analýze. Informace o provozu budou sbírány z jednotlivých komponent BeeeOn brány, které jsou zpřístupňovány pomocí návrhového vzoru *Observer*. Z tohoto důvodu bude kolektor vždy přímou součástí BeeeOn brány. Návrh struktury kolektoru je na obrázku 3.2



Obrázek 3.2: Návrh kolektoru provozních dat

*AbstractCollector*, je třída vytvořená v projektu BeeeOn, které shromažďuje všechny dostupné události. V současné době jsou k dispozici následující události:

- **onExport**: reprezentuje hodnoty dat přicházejících ze senzorů, která jsou odeslána vždy po jejich příjmu
- **onDriverStats**: v pravidelných intervalech generuje statistiky o Z-Wave síti dostupné na komunikačním rozhraní
- **onNodeStats**: periodicky získává informace o jednotlivých prvcích Z-Wave sítě, které jsou dostupné pomocí OpenZWave knihovny
- **onHciStats**: v definovaných intervalech získává z komunikačního rozhraní pro BLE statistiky o provozu sítě
- **onDispatch**: umožňuje získávat zadané uživatelské příkazy

Vytvořený kolektor bude reprezentován třídou *NemeaCollector*, která bude potomkem třídy *AbstractCollector* a bude implementovat všechny její členské funkce pro zachytávání událostí. Pro každou členskou funkci bude vytvořen setter, který při spuštění brány provede počáteční nastavení parametrů pro obsluhu dané události. Vstupním parametrem bude řetězec určující název výstupního rozhraní, který bude specifikovaný v konfiguračním souboru BeeOn brány. Po nastavení počátečních parametrů jako je název výstupního rozhraní a seznam UniRec polí se zavolá členská funkce *initInterface*, jejímž cílem bude inicializace veškerých struktur vyžadovaných NEMEA frameworkem. Jako vstupní parametr bude přijímán objekt *EventMetaData* udržující veškeré informace potřebné k odesílání získaných dat.

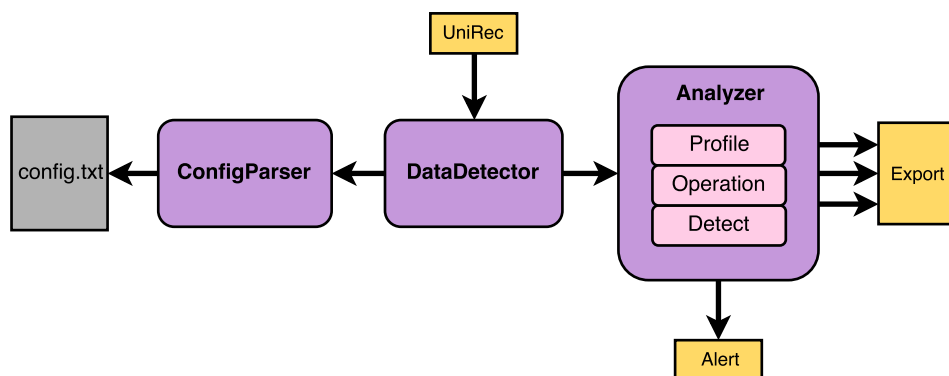
Instance objektu *EventMetaData* bude vytvořena jako členská proměnná třídy *NemeaCollector* pro každou členskou funkci události. Díky tomu bude možné každou událost zpracovávat nezávisle dle definovaných parametrů.

Definované konstruktory a destruktory budou sloužit jen pro vytvoření a destrukci potřebných struktur. Tímto bude zajištěn programovací idiom RAI (Resource Acquisition Is Initialization).

Veškeré odesílané údaje budou rozšířeny o časovou značku a číselný identifikátor, aby mohl detektor přesně určit původ případné anomálie.

### 3.3 Detektor

Detektor bude hlavní komponentou celého řešení, protože bude vyhodnocovat získaná data. Vzhledem k zadaným požadavkům musí být návrh proveden tak, aby bylo umožněno budoucí rozšiřování. Návrh architektury se nachází na obrázku 3.3



Obrázek 3.3: Architektura detektoru

Komponenta *DataDetector* bude mít jedno vstupní rozhraní, kterým bude přijímat data z kolektoru. V případě potřeby zpracování více událostí z kolek-

toru lze využít již existující modul *Merger* z NEMEA repozitáře, který dokáže sloučit více vstupů do jednoho konsolidovaného výstupu.

Dále se v rámci třídy *DataDetector* bude volat *ConfigParser*, který bude mít na starosti zpracování konfiguračního souboru. Konfigurační soubor bude obsahovat seznam parametrů pro každé UniRec pole, na základě kterých bude vytvořena instance třídy *Analyzer* pro analýzu dat.

Po provedení inicializace všech modulů bude *DataDetector* pouze přijímat data z kolektoru a předávat je ke zpracování komponentě *Analyzer*. Tato komponenta bude obsahovat tři hlavní části:

- **Profile**

Bude obsahovat členské funkce, které se budou starat o připravení výchozího profilu sítě. Vytvořený profil bude složen z: průměru, klouzavého průměru, rozptylu a mediánu. Tyto hodnoty budou následně použity při porovnávání s aktuálním provozem v rámci detekčních metod. V případě potřeby bude možné profil rozšířit o další hodnoty.

- **Detect**

Bude sdružovat všechny členské funkce určené pro detekci anomálií v získaných datech. Každá členská funkce bude reprezentovat jedno kritérium a bude možné budoucí rozšíření o další potřebné metody.

- **Operation**

Cílem této části bude udržování veškerých použitých struktur. Bude se jednat o struktury pro konfiguraci, zpracovávaná data a odesílání získaných výsledků. Odesílat lze informaci o nalezené hrozbě, pro kterou se používá jedno společné rozhraní, nebo pravidelně exportovat definované části aktuálního profilu sítě pro účely dalšího zpracování.

Díky konfiguraci organizované po jednotlivých UniRec polích je možné nastavit chování výše popsanych částí nezávisle na sobě pro každé pole.

### 3.3.1 Konfigurace modulu

Jedním z funkčních požadavků je možnost konfigurace způsobu detekce, která se tak bude moci přizpůsobit cílovým podmínkám. Každá instance detektoru bude mít jeden konfigurační soubor, který bude popisovat jednotlivá UniRec pole určená pro analýzu.

Konfigurace bude v textové podobě ve formátu klíč a seznam hodnot. Textová podoba je velmi vhodná pro fázi ladění a vývoje nástroje, protože jsou možné manuální úpravy. Cílem ovšem je generovat konfiguraci pomocí externího nástroje, který by mohl být součástí řídicí vrstvy IoT brány. Generátor nastavení bude velmi vhodný pro koncové uživatele, protože si budou moci přehledně vybrat své parametry bez znalosti všech závislostí, které textová

konfigurace ani modul pro zpracování nebude hlídat. Externí nástroj pro definování nastavení nebude součástí vytvořeného řešení.

Klíčem bude vždy název UniRec pole, který se bude nacházet v příchozích datech. Za tímto klíčem se může nacházet volitelně číselný identifikátor senzoru, který se musí shodovat s identifikátorem odesílaným z kolektoru. Důsledkem toho bude možnost detekční parametry definovat i pro jednotlivá zařízení pokud se jich pod jedním UniRec názvem bude nacházet více. V případě neuvedení identifikátoru se jako výchozí hodnota použije číslo nula. Takto definovaná hlavička bude oddělena dvojtečkou, za kterou bude sekvence parametrů oddělených středníkem . Každý nový záznam s popisem přicházejících dat se bude nacházet na samostatném řádku. Řádky prázdné a začínající znakem # budou považovány za komentář.

Definované parametry budou rozděleny do následujících hlavních skupin:

- **General**

Tato skupina bude obsahovat obecné volby pro definování časové řady. Pomocí číselných parametrů bude možné určit velikost řady, délku učicí fáze, během které bude vytvořen profil sítě a její délka musí být stejná nebo delší než velikost časové řady, a rozsah ignorovací části, která bude vhodná k odfiltrování počáteční výměny zpráv při navazování spojení.

Dalším parametrem bude způsob ukládání dat do časové řady. K dispozici budou dva režimy. Prvním z nich bude *simple*, který přijatou hodnotu pouze uloží. Druhý bude *delta*, který bude vkládat rozdíly dvou po sobě jdoucích hodnot. Toto bude vhodné zejména u časových značek nebo počtu odeslaných zpráv, protože bude vhodnější udržovat jednotlivé přírůstky než pouhé absolutní hodnoty.

Poslední volbou budou číselné hodnoty určující rozmezí mezi pravidelnými kontrolami přijetí a změny dat a číselný parametr definující periodu pro exportování vybraných parametrů. Nastavení těchto položek bude volitelné. Jejich nepoužití bude označeno znakem -.

- **Profile**

Zápis této skupiny bude identifikovaný klíčovým slovem *profile*, který má uvnitř kulatých závorek vyjmenované jednotlivé položky, které budou součástí vytvořeného profilu. Ve vytvořeném řešení budou k dispozici následující volby: *average* (průměr), *variance* (rozptyl), *median* (medián) a *cum\_average* (klouzavý průměr). Součástí profilu bude vždy alespoň jedna položka. Použité části a jejich počet bude vždy záviset na uživateli.

- **Profile Values**

Pro každou položku definovanou v rámci profilu musí být uveden seznam jejich detekčních parametrů. Zápis bude probíhat pomocí klíčového slova

z profilu, které bude mít jednotlivé hodnoty specifikované uvnitř kulatých závorek. Zde bude možné určit minimální a maximální soft a hard limity, kterých daná položka aktuálního profilu může dosahovat. Pro soft limit bude možné určit povolenou délku, po kterou může být soft limit překročen (grace period).

Dalšími parametry budou číselné hodnoty určující hranice pro minimální a maximální velikost změny sledované položky. Všechny ostatní argumenty nebudou mít žádný význam pro konfiguraci, ale budou sloužit jako proměnné pro ukládání dočasných hodnot.

Zadání veškerých hodnot nebude nutné a tudíž bude záležet na uživateli, které způsoby detekce bude chtít využít. V případě nedefinování některého parametru bude zapsán znak -.

#### • Export

Poslední nastavitelnou možností bude export vybraných dat z profilu. Zápis bude probíhat pomocí klíčového slova *export*, který v kulatých závorkách bude mít seznam položek z profilu. Zároveň v případě využití této možnosti bude nutné ve skupině *general* určit časovou periodu odesílání. Názvy výstupních *libtrap* rozhraní budou odvozené od klíčů, které se nacházejí před dvojtečkou v konfiguraci.

Příklad záznamu s konfigurací je na obrázku 3.4. Jednotlivé barvy označují výše definované skupiny.

Nastavení formou souboru bude jedinou volbou pro přizpůsobení detekce. Dále při spuštění programu bude očekáván povinný přepínač *-i*, který je vyžadován knihovnou *libtrap* a bude definovat název vstupního rozhraní pro příchozí data a název výstupního rozhraní pro detekované incidenty. Jako nepovinný přepínač bude možná zadat *-v*, jehož výsledkem bude zapnutí ladících výpisů. Tato volba bude umožňovat až tři úrovně podrobnosti výpisů. Tyto úrovně budou určeny počtem znaků *v*.

```
TIME: 5;12;3;delta;20;30; profile(average,variance,); average(0,6,-1,10,5,2,0.2,0,0); variance(-,-,-,-,0.5,5,0,0); export(variance,);
```

Obrázek 3.4: Popis konfiguračních parametrů pro jeden záznam

### 3.3.2 Použité struktury

Velmi důležitou částí návrhu bude určení formátu použitých struktur, které budou udržovat veškeré údaje nutné pro analýzu. V rámci detektoru budou potřeba udržovat informace pro následující položky:

- **Konfigurační parametry**

Tato struktura bude hierarchicky ukládat veškeré hodnoty z konfiguračního souboru a také údaje nutné k provedení analýzy. V první úrovni se budou nacházet názvy jednotlivých UniRec polí, které budou fungovat jako klíče. V druhé vrstvě bude seznam identifikátorů odlišujících senzory skryté pod stejným klíčem. Třetí úroveň bude vždy obsahovat stejnou sekvenci podklíčů, které budou určovat typ uchovávaných dat. Bude se jednat o jednotlivé skupiny z konfiguračního souboru 3.3.1, které budou označeny stejnojmennými identifikátory. Hodnotou těchto skupin budou specifikované parametry detekce dle zapsané konfigurace. Dále zde budou podklíče s názvem *metaProfile* a *metaData*, které budou mít stejné hodnoty. Tyto hodnoty budou obsahovat vypočtené údaje z příslušné časové řady a zároveň budou umožňovat ukládání dat, která budou nutná k provedení výpočtu a vkládání nových prvků. Rozdíl mezi *metaProfile* a *metaData* bude typ profilu, pro který budou určeny. Podklíč *metaProfile* bude uchovávat údaje pro základní profil, který bude vytvořen během učící fáze a bude sloužit pro určení anomálie. Zatímco hodnoty v *metaData* budou vztaženy k aktuálnímu profilu sítě, který se mění s každým novým prvkem v časové řadě.

- **Senzorová data**

Cílem této struktury bude hierarchicky ukládat přicházející data z kolektoru do časových řad. V první úrovni se jako v předchozím případě budou nacházet názvy jednotlivých UniRec polí. Protože jedno UniRec pole může reprezentovat data z více různých senzorů nebo bran, tak se na další úrovni nachází identifikátor zdroje dat. Předpokládá se, že každá použitá brána a senzor bude mít v rámci jedné instance detektoru unikátní identifikátory. Na třetí úrovni se bude vyskytovat definovaná časová řada, která uloží data podle zadané konfigurace.

- **Nastavení pro export**

Parametry pro export budou uchovány ve struktuře s formátem klíč a hodnota. Klíčem bude pořadové číslo výstupního rozhraní, které bude odpovídat konkrétnímu UniRec poli. Hodnotou bude sekvence názvů určující jednotlivé části profilu, které se budou exportovat. Tyto názvy budou definované v konfiguračním souboru.

#### 3.3.3 Způsob analýzy

Veškerá analýza dat bude prováděna pomocí časových řad, které jsou vhodné pro statistický typ dat a metodu anomálií. Zároveň parametry řad budou nastavitelné, aby bylo možné se přizpůsobit pro danou datovou položku, ale i cílovou bránu, na které může být spuštěn detektor. Způsob zpracování dat lze rozdělit do následujících fází:

- **Inicializace**

V rámci této sekce budou načteny všechny konfigurační položky a inicializovány veškeré struktury nutné pro běh celého řešení. Následně bude program očekávat příchozí data na jediném vstupním rozhraní.

- **Učení**

Po prvotním přijetí hodnot z kolektoru bude probíhat stanovení vzorového profilu sítě pro položky, které budou popsány v konfiguračním souboru. V rámci této periody se do časové řady budou vkládat přicházející data a s každým novým prvkem bude prováděn výpočet všech částí profilu, které budou určeny konfiguračním souborem. Pokud bude délka učicí fáze delší než velikost časové řady, tak budou všechny prvky posunuty, a tím nejstarší z nich uvolní místo pro nový.

- **Detekce**

Jakmile bude stanoven vzorový profil sítě, tak se zahájí analýza provozu pro určení anomálií. S každým novým příchozím prvkem bude z příslušné časové řady vypočítáván aktuální profil sítě, který budou používat detekční metody.

- **Vyhodnocení**

Výsledky detekčních metod budou sloužit k určení stavu provozu. Pokud budou data vyhodnocena jako bezpečná, tak nebude provedena žádná akce a přejde se k obsluze další přijaté hodnoty z kolektoru. V případě nalezení anomálie se vytvoří zpráva s informacemi popisující událost, která bude odeslána výstupním rozhraním určeným pro detekované hrozby.

## 3.4 Multiplexor a demultiplexor

Hlavním cílem těchto dvou modulů bude ulehčit nasazení analýzy dat v případě fyzicky odděleného kolektoru a detektoru. Bez použití těchto modulů bude nutné na kolektoru zabezpečit přenos dat z každého výstupního rozhraní, což zvýší nároky na výkon brány a zároveň bude vyžadováno více konfigurace. Díky multiplexoru bude možné spojit několik různých *libtrap* komunikačních rozhraní do jednoho, které lze demultiplexorem zase zpětně rozdělit na cílovém zařízení. Tímto bude pro export dat z brány nutné zabezpečit pouze jedno rozhraní.

Multiplexor bude vícevláknový NEMEA modul, kde každé vlákno bude naslouchat na specifikovaném vstupním rozhraní. Po přijetí nového formátu zprávy se nejprve odešle *hello* zpráva, která bude obsahovat popis nového formátu. V případě přijetí zprávy se stejným formátem bude tato zpráva rovnou přeposlána. Oba typy zpráv budou mít stejnou hlavičku s položkami: identifikátor zprávy, číslo vstupního rozhraní a typ formátu. Ostatní data nutná pro přenos budou zapouzdřena v záhlaví. Díky stejnému formátu zprávy bude možné veškerá data zasílat jedním společným rozhraním.

Demultiplexor bude jednovláknový modul, který bude přijímat na jednom vstupním rozhraní zprávy od multiplexoru. Po přijetí dat provede jejich zpracování a na základě identifikátoru provede příslušnou akci. V případě typu



*hello* se upraví nový formát pro specifikované rozhraní. V druhém případě se jen přepošlou přijatá data příslušným výstupem.

### 3.5 Scénáře útoků

Na základě vypracované analýzy a provedených experimentů s protokoly Z-Wave a BLE byly navrženy následující scénáře, které mohou v reálném provozování IoT sítě nastat a identifikovat zneužití zranitelnosti díky vzniku anomálního provozu. V popsanych scénářích se bude vyskytovat jen nezbytně nutná množina informací, kterou bude nutné sledovat. Popis všech dostupných dat se nachází v příloze.

#### 1. Periodicita dat

Velké množství koncových prvků v senzorové vrstvě se chová periodicky. Kvůli úspoře napájecí energie jsou senzory často v režimu spánku a jen v určitých časových okamžicích se probudí, aby odeslaly potřebné údaje. Dále lze využít pravidelnosti BeeeOn brány, která statistiky o jednotlivých komunikačních protokolech exportuje ve stálých časových intervalech.

Tento scénář se bude soustředit na analýzu času příchozích dat, aby určil případné nestandardní chování. Díky tomu bude možné odhalit vzniklé problémy na bráně, které vedou k nepravidelnému odesílání informací o provozu. V případě, že bude použito čidlo s periodickým chodem, bude možné zjistit změnu jeho chování nebo jeho úplné odpojení.

Určení času příchozích dat bude snadné, protože každý přijatý záznam bude obsahovat datovou položku s časovou značkou.

#### 2. Množství přenášených zpráv

Velkou hrozbou, kterou představuje nástup IoT jsou nejrůznější varianty DoS útoků. Konkrétně nejobávanějším typem je pravděpodobně zahlcení cílové komunikační linky, protože v síti bude dostupné obrovské množství prvků, které útočník může v případě napadení využít. Zároveň nejlepším místem, kde tyto útoky zastavit, je zdroj jejich vzniku. Tento scénář může také identifikovat připojení dalšího zařízení.

Cílem bude identifikovat neobvyklý nárůst nebo pokles přenášených dat v porovnání s naučeným vzorovým profilem sítě.

Výhodou jsou Z-Wave sítě, protože nabízí statistiky pro celou síť i pro jednotlivé senzory. Položka *SOAFCount* určuje celkový počet detekovaných zpráv ve vytvořené topologii. Při analýze jednotlivých čidel bude možné sledovat čítače *receivedCount* a *sentCount*. První čítač reprezentuje počet zpráv od konkrétního čidla, který byl přijat na bráně. Druhá hodnota určuje počet odeslaných zpráv danému senzoru.

### 3. NÁVRH

---

BLE nabízí pouze celkové informace, kterých navíc není tolik. V rámci této detekce bude vhodné použít položky *rxBytes* a *txBytes*, které odpovídají počtu přijatých a odeslaných bytů.

#### 3. Limity senzorových hodnot

Tento případ užití se zaměřuje přímo na získávané hodnoty z čidel. Pro tyto hodnoty se většinou dá určit jejich minimální a maximální hodnota s rychlostí růstu. Pokud se vývoj hodnot nebude shodovat s očekáváním, může to znamenat poruchu čidla nebo neočekávanou událost, jejíž příčinou může být i útok.

Možnost této detekce bude záviset na události *onExport*, která exportuje všechna získaná data nezávisle na použitém protokolu.

#### 4. Kvalita přenosového kanálu

Zejména u bezdrátových komunikačních protokolů je velmi důležitá kvalita přenosového kanálu, který může být případným útočníkem zarušen. V situaci, že tento typ útoku nastane, budou detekční metody očekávat zhoršení statistik komunikace.

Knihovna OpenZWave nabízí pro každé čidlo informace o hodnotě RTT (Round-Trip Time). Z pohledu celé sítě bude možné zjistit počet zahozených zpráv (položka *dropped*), počet opakovaných odeslání zpráv (položka *retries*) nebo počet zpráv z poškozeným kontrolním součtem (položka *badChecksum*).

V případě BLE lze využít položky *rxErrors* a *txErrors*, které reprezentují počet přijatých a odeslaných chyb.

#### 5. Konektivita

Tento případ navazuje na předchozí 4. Kvalita připojení může být zhoršena natolik, že dojde ke ztrátě konektivity. Případně spojení může být ztraceno pokud je senzor odcizen.

Analýza může probíhat na základě parametrů specifikovaných v předchozím scénáři. Nebo lze ztrátu konektivity se senzorem identifikovat tím, že se provozní statistiky nezměnily vůbec nebo menším způsobem než bylo obvyklé. Pro tento typ detekce mohou být využity stejné položky jako ve druhém případě užití 2.

Všechny definované situace budou zároveň sloužit k otestování detekčního systému, který by je měl všechny odhalit.

## Realizace

Obsahem kapitoly je popis realizace nejzajímavějších částí vytvořeného programu. Pro zajištění efektivity a snadné přenositelnosti byla veškeré implementace provedena v jazyce C++, ve kterém je napsána BeeeOn brána. Detekční systém přináší pouze jednu novou závislost a tou je NEMEA framework. Ostatní použité knihovny se shodují s již použitými v BeeeOn bráně.

### 4.1 Integrace kolektoru

Vytvořený kolektor provozních dat je přímou součástí projektu BeeeOn, a proto jej bylo nutné integrovat do spouštěcího procesu brány. Pro zajištění společné kompilace byly do souborů *CMakeLists.txt* přidány cesty ke zdrojovým souborům kolektoru a závislosti na knihovny z NEMEA frameworku. K určení způsobu spouštění jednotlivých komponent používá BeeeOn soubor *factory.xml*. V tomto souboru byla vytvořena nová komponenta s názvem *collector*, která byla následně přidána pod označením *listener* do ostatních komponent. Toto nastavení umožňuje přijímání definovaných událostí v rámci návrhového vzoru *Observer*. Samotná komponenta obsahuje ve svém popisu seznam jmen událostí, kterým přiřazuje pojmenování výstupního *libtrap* rozhraní. Formát výstupu je vždy *event-<názevUdálosti>*.

Vytvořené názvy událostí jsou při spouštění programu pomocí C++ reflexe předány třídě kolektoru *NemeaCollector*. Tato třída používá již vytvořené makro *BEEEON\_OBJECT\_TEXT*, které pro definovaný seznam událostí volá příslušné členské funkce. Každá událost má členskou funkci s názvem ve formátu *set<názevUdálosti>* přijímající jeden vstupní parametr typu string s názvem výstupního rozhraní. V rámci volání se do instance třídy *EventMetaData* nastaví specifické hodnoty členských atributů dané události. Následuje volání členské funkce *initInterface(EventMetaData)*, která přijímá vytvořenou instanci třídy *EventMetaData* a jednotně inicializuje všechny potřebné struktury pro odesílání dat.

Po úspěšném nastavení všech částí se už jen v rámci návrhového vzoru *Observer* volají členské funkce událostí, které jsou definované ve třídě *AbstractCollector* a implementované ve třídě *NemeaCollector*. Získané informace jsou vkládány do UniRec zprávy a odeslány výstupním rozhraním.

## 4.2 Mux a Demux

Modul *Mux* očekává na vstupu přepínač *-i*, který ve formě řetězce určuje dostupná rozhraní zajištěné knihovnou *libtrap*. Poslední identifikátor v řetězci označuje jméno výstupu. Druhým parametrem je *-n*, který odpovídá počtu vstupních rozhraní. Při spuštění se pro každý vstup pomocí knihovny OpenMP vytvoří samostatné vlákno. Jelikož je veškerý provoz odesíláný jedním společným rozhraním, bylo nutné funkci *trap\_ctx\_send* vložit do kritické sekce, protože pracuje se sdílenými strukturami pro všechny vlákna.

Společné spojení mezi moduly vytvořenými *Mux* a *Demux* používá nastavený typ *TRAP\_FMT\_RAW*, který umožňuje posílat zprávy ve vlastně definovaném formátu. Hlavička vytvořeného formátu obsahuje: identifikátor druhu zprávy, číslo rozhraní a typ formátu. Obsah přijatých zpráv je zapouzdřen do záhlaví. *Mux* při každém přijetí dat kontroluje návratový kód funkce *trap\_ctx\_recv*, který identifikuje nový formát přijatých zpráv. Pokud dojde k detekování změny, tak se pošle *hello* zpráva s upraveným popisem rozhraní. V ostatních případech se jen přeposílají zapouzdřená data.

Modul *Demux* vyžaduje stejné přepínače jako *Mux*. Jediným rozdílem je, že název společného rozhraní, které má *Mux* na posledním místě, musí být zde uveden jako první. Důvodem je, že knihovna *libtrap* zpracovává nejprve vstupní a pak výstupní rozhraní.

## 4.3 Zpracování zadaných parametrů

Načtení konfiguračního souboru má na starost třída *ConfigParser*, která neprovádí žádnou kontrolu vstupních dat, protože se již od návrhu předpokládá, že konfigurace bude generována odlišným programem, který zajistí správnost parametrů. Jediný konstruktor třídy *ConfigParser* očekává jako parametr řetězec s cestou k cílovému souboru. Během vytváření objektu jsou postupně zpracovávány jednotlivé řádky zadaného souboru. Zároveň budou inicializovány pole pro uchovávání vypočtených profilů. Jejich délka je specifikována preprocesorovou direktivou *#define DYNAMIC*.

Pokud byly zadány parametry pro pravidelný export dat, tak při inicializaci potřebných struktur zavolá funkce s názvem *initExportInterfaces*, která vytvoří příslušná výstupní rozhraní. Jejich název je vždy vygenerován v následujícím formátu: *u:export-<názevKlíče><idKlíče>*.

#### 4.4 Výpočet profilu

#### 4.5 Detekční funkce



# Testování

V této kapitole je popsán postup testování vytvořeného detekčního řešení. Nejprve je popsáno testovací prostředí. Dále jsou otestovány obě možné metody nasazení. V závěru jsou ověřeny jednotlivé případy hrozeb z kapitoly 3.5 Scénáře útoků a také jsou prověřeny ostatní funkcionality detektoru.

## 5.1 Testovací prostředí

Veškeré provedené testy proběhly na lokálním počítači s operačním systémem Ubuntu 16.04, na kterém zároveň proběhl vývoj nástroje. Systémové prostředky dostupné pro testování byly: 4 jádra CPU, 8 GB RAM a 17 GB SSD disk.

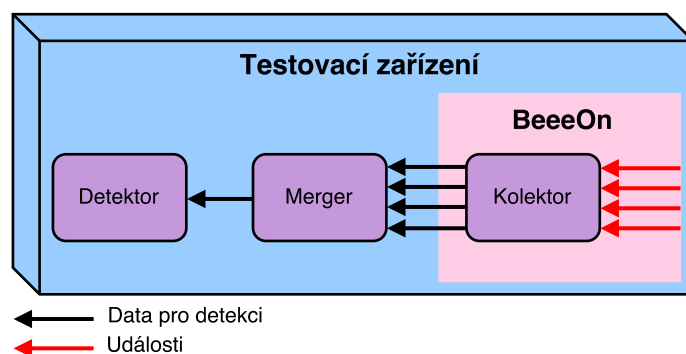
Na počítači byl nasazen systém NEMEA a IoT brána BeeeOn, která byla rozšířena o vytvořený detekční nástroj. Pro generování senzorových dat byly použity následující zařízení: BLE teplotní senzor (BeeWi), Z-Wave zásuvky (POPP a Fibaro) a virtuální senzory, které jsou dostupné pro testování na BeeeOn bráně. Sensorová data přijímal testovací počítač, který měl připojený USB Z-Wave dongl (Aeotec) a integrované BLE rozhraní.

## 5.2 Způsoby nasazení

Jako první byly testovány možnosti nasazení vytvořeného nástroje, který je možné používat v následujících režimech:

- **Lokální režim**

V lokální variantě byl použit pouze modul detektoru a kolektoru. Aby se nemusela pro každou událost spouštět nová instance detektoru, byl využit již existující NEMEA modul *Merger*, který dokáže jednotlivé UniRec zprávy spojit do jedné konsolidované. Způsob nasazení je na obrázku 5.1.

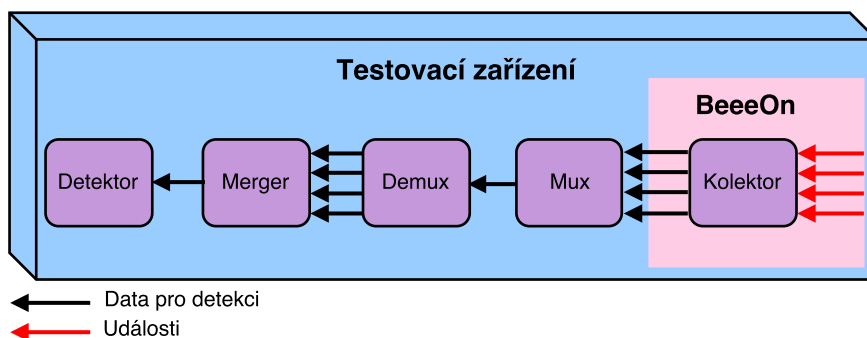


Obrázek 5.1: Lokální nasazení

Kolektor odesílá vždy dostupné události specifikovanými výstupními rozhraními dle konfigurace brány. Exporovaná data přijímá NEMEA modul *Merger*, který je spojuje a předává detektoru.

### • Oddělený režim

Druhý způsob nasazení byl také otestován na lokálním zařízení, protože lze využít lokálních soketů, které nabízí NEMEA framework. Způsob zavedení je na obrázku 5.2.



Obrázek 5.2: Oddělené nasazení

Složení komponent vychází z prvního případu, který byl rozšířen o moduly Mux a Demux, které umí spojit a rozdělit přicházející komunikaci.

Cílem testů bylo, aby detektor obdržel všechny odeslané zprávy z kolektoru. Ověření bylo provedeno spuštěním detektoru s přepínačem *-vv*, který zapne ladící výpisy druhé úrovně pro zobrazení přijatých UniRec polí. Výsledky obou případů byly úspěšné a všechna data byla přijata.



## 5.3 Detekce scénářů útoků

Velmi důležitou částí bylo otestování definovaných anomálií, které mohou reprezentovat skutečný útok na síť. Tato sekce popisuje testy pro jednotlivé scénáře.

### 1. Periodicita dat

Tento scénář nebylo nutné dělit na jednotlivé protokoly, protože popisuje obecné chování připojených senzorů bez ohledu na technologii. Cílem bylo odhalit provoz, který porušuje očekávaný periodický průběh. V rámci detekce nebyly potřebné naučené profily sítě, ale využívalo se parametrů ze skupiny *general* umožňující periodické kontroly.

První test byl určený na odhalení nepravidelného přijetí dat. Pro ověření byla použita událost *onExport*, která poskytuje hodnoty získané ze senzorů, a v konfiguračním souboru byla aktivována pravidelná kontrola dat každých 8 sekund.

Výsledek testu byl úspěšný. Pokud nebyla obdržena žádná data déle než 8 sekund, tak byly odeslány informace o incidentu.

Několik událostí poskytovaných branou BeeeOn jsou čistě periodické a po uplynutí definovaného času vždy odešlou dostupné statistiky. Druhý test byl zaměřen na odhalení případu, kdy se přijímané datové položky nemění, což může reprezentovat odpojení nebo ztrátu čidla. Pro otestování byla zvolena událost *onHciStats*, které získává informace o provozu BLE sítě. Dále byla v konfiguračním souboru nastavena pravidelná kontrola dat na 7 sekund a maximálně 5 po sobě přijatých hodnot mohlo být stejných.

Výsledek byl pozitivní, protože při přijetí více než pěti stejných po sobě jdoucích hodnot byla detekována anomálie.

### 2. Množství přenášených zpráv

V případě protokolu Z-Wave byly pro nasimulování anomálií použity dvě vzdáleně ovladatelné zásuvky. Obě mají k dispozici ovládací tlačítko pro vypnutí a zapnutí zásuvky. Tímto způsobem byly generovány nové zprávy. Cílem tohoto scénáře bylo odhalit neočekávaný nárůst dat, proto byla zvolena detekční metoda, která hlídá tyto změny. V rámci testu byly do profilu vloženy všechny dostupné položky. Limitem pro ohlášení incidentu byl pětinašobný nárůst provozu. Délka časové řady byla nastavena na deset prvků a prvních jedenáct přijatých zpráv bylo ignorováno, protože během nich bylo navazováno spojení.

První detekce byla zaměřena na položku *SOAFCount*, která určuje celkový počet detekovaných zpráv. Druhý scénář sledoval hodnotu *receivedCount*, která reprezentuje počet přijatých zpráv od konkrétního čidla.

Pro BLE byla časová řada zkrácena na pět prvků, žádné zprávy nebyly ignorovány a limitem pro určení anomálie byl dvojnásobný nárůst dat. Test byl proveden pro položku *rxBytes*, která určuje množství obdržených zpráv. Data byly vyčítány skriptem, který v čase měnil četnost zaslaných požadavků o data.

Výsledky testů byly úspěšné a každá změna provozu byla odhalena. Průběh všech testovaných případů byl velmi podobný a lišil se jen typ události a způsob generování dat. Chování jednotlivých položek profilu velmi ovlivňovala délka časové řady, která určovala paměťové okno. Nejcitlivější položkou na změnu byl rozptyl, který výrazně zvyšoval svou hodnotu při vložení vyššího čísla. K častým výchýlkám docházelo i v případě průměru. Méně frekventované změny nastávaly u mediánu, který nejvíce využíval délky časové řady. Nejmenších odlišností dosahoval klouzavý průměr, protože ve své hodnotě obsahuje i data mimo aktuální časové okno.

### 3. Limity senzorových hodnot

Pro tento scénář byla použita data vygerovaná virtuálními senzory, které jsou dostupné v rámci brány BeeeOn. Do konfiguračního profilu byly vloženy všechny položky, pro které byly specifikovány parametry pro očekávané soft a hard limity. Cílem scénáře bylo odhalení změn v aktuálním profilu provozu, které porušují předepsané limity.

Výsledek detekce splnil očekávání a nepovolené změny byly úspěšně detekovány. Chování jednotlivých položek profilu se shodovalo s předchozím scénářem.

### 4. Kvalita přenosového kanálu

Cílem tohoto případu užití bylo detekování změny kvality přenosového kanálu. Pro Z-Wave jsou k tomuto účelu vhodné položky *lastResponseRTT* a *dropped*. Údaj *lastResponseRTT* je součástí události *onNodeStats* a *dropped* je obsažen v *onDriverStats*. Z tohoto důvodu byl při generování dat použit NEMEA modul *Merger*, který spojil dvě různé události do jedné. V konfiguračním souboru bylo nutné popsat obě hodnoty. V případě *lastResponseRTT* byla jako anomálie označena hodnota průměru časového okna přesahující dvojnásobek vzorového profilu a nebo byla nižší než polovina původního průměru. Pro položku *dropped* bylo jako incident považováno libovolné zvětšení mediánu. Proto byla velikost časového okna rovna jedné a limit hard limitu byl nastaven na jedna.

Jelikož došlo ke spojení dvou různých událostí, kde *onNodeStats* umožňuje získat informace pro jednotlivé senzory a *onDriverStats* pro celou síť, tak výsledkem byla událost poskytující informace pro každý koncový

prvek. Z tohoto důvodu musel být v konfiguraci uveden identifikátor příslušných zařízení určených k analýze.

Výstupy detekce se shodovaly s předpoklady, a proto byl výsledek testu úspěšný. V současné době se zatím nepodařilo připravit zkušební prostředí, ve které by bylo možné otestovat rušení kanálu, a tím ovlivnit hodnotu *lastResponseRTT*. Pro účely otestování detekce však byla postačující ruční úprava dat v souboru se zachyceným provozem.

V případě BLE by se využily položky *rxErrors* nebo *txErrors* a nastavení detekce by bylo shodné s *lastResponseRTT*. Vzhledem k tomu, že by došlo jen ke změně názvu klíče v konfiguračním souboru, tento test nebyl proveden.

## 5. Konektivita

Pro ověření posledního scénáře lze využít poznatků z předchozích testů. Ztráta konektivity může být způsobena neobráněním očekávané události a to lze detekovat pomocí periodicity dat. Druhým způsobem je výrazné zhoršení statistik přenosového kanálu, které byly zpracovány v předchozím případě užití.

Ověření definovaných scénářů umožnilo otestování správné funkcionality vytvořeného detekčního systému a zároveň byla sestavena množina informací, kterou je nutné sledovat k získání přehledu o stavu sítě.

Vygenerovaný provoz, na kterém byly jednotlivé případy užití otestovány je uložen i s nalezenými anomáliemi a konfiguračním souborem na příloženém CD. V případě testu periodicity dat nebylo možné záznam komunikace uložit, protože použitá detekce využívá aktuální čas. Z tohoto důvodu byl do souboru místo záznamu uložen popis, jak takový provoz vygenerovat.

## 5.4 Test exportu dat

Identifikované scénáře útoků otestovaly většinu funkcí detekčního systému. Vytvořený nástroj ovšem poskytuje i možnost pravidelného exportu definovaných položek aktuálního profilu, a tím vytváří informace pro další pokročilejší detekce.

Sada testů ověřila, že vypočítané části profilu lze spolehlivě exportovat. Výsledek byl tedy pozitivní.

## 5.5 Měření položek profilu



---

## **Závěr**



---

## Literatura

- [1] Statista: Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025. Statista. [online], 2016 [cit. 2018-01-14]. Dostupné z: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>
- [2] Milici, S.; Lázaro, A.; Villarino, R.; aj.: Wireless Wearable Magnetometer-Based Sensor for Sleep Quality Monitoring. *IEEE Sensors Journal*, 2018: s. 2145–2152.
- [3] Whitehead, D. E.; Owens, K.; Gammel, D.; aj.: Ukraine cyber-induced power outage: Analysis and practical mitigation strategies. In *2017 70th Annual Conference for Protective Relay Engineers (CPRE)*, 2017, s. 1–8.
- [4] Statista: Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Cisco. [online], 2015 [cit. 2018-01-15]. Dostupné z: [https://www.cisco.com/c/dam/en\\_us/solutions/trends/iot/docs/computing-overview.pdf](https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf)
- [5] BeeeOn: Main Page. beeeon.org. [online], 2017 [cit. 2018-02-06]. Dostupné z: <https://beeeon.org>
- [6] Zhao, K.; Ge, L.: A Survey on the Internet of Things Security. In *2013 Ninth International Conference on Computational Intelligence and Security*, 2013, s. 663–667.
- [7] Pacheco, J.; Hariri, S.: IoT Security Framework for Smart Cyber Infrastructures. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self\* Systems (FAS\*W)*, 2016, s. 242–247.
- [8] OASIS: MQTT Version 3.1.1 Plus Errata 01. OASIS. [online], 2015 [cit. 2018-03-08]. Dostupné z: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html>

- [9] dc square: MQTT Essentials Wrap-Up. HiveMQ. [online], 2018 [cit. 2018-03-08]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-wrap-up>
- [10] dc square: MQTT Security Fundamentals. HiveMQ. [online], 2018 [cit. 2018-03-08]. Dostupné z: <https://www.hivemq.com/mqtt-security-fundamentals/>
- [11] Shelby, Z.; Hartke, K.; Bormann, C.: The Constrained Application Protocol (CoAP). RFC7252. [online], 2014 [cit. 2018-03-08]. Dostupné z: <https://tools.ietf.org/html/rfc7252>
- [12] Alliance, Z.-W.: Z-Wave Plus Certification. Z-Wave Alliance. [online], 2018 [cit. 2018-03-17]. Dostupné z: [https://z-wavealliance.org/z-wave\\_plus\\_certification/](https://z-wavealliance.org/z-wave_plus_certification/)
- [13] OpenZWave: OpenZWave. OpenZWave. [online], 2018 [cit. 2018-03-17]. Dostupné z: <http://www.openzwave.net/>
- [14] Krejčí, R.; Hujňák, O.; Švepeš, M.: Security survey of the IoT wireless protocols. In *2017 25th Telecommunication Forum (TELFOR)*, 2017.
- [15] Designs, S.: Introduction to Z-Wave Specifications. ZWave. [online], 2016 [cit. 2018-03-18]. Dostupné z: <http://zwavepublic.com/specifications>
- [16] Fouladi, B.; Ghanoun, S.: Security Evaluation of the Z-Wave Wireless Protocol. In *BlackHat Conference, Las Vegas, NV, USA*, 2013 [cit. 2018-03-18].
- [17] Jasek, S.: Gattacking Bluetooth Smart Devices. SecuRing. [online], 2016 [cit. 2018-03-18]. Dostupné z: <http://gattack.io/whitepaper.pdf>
- [18] Fouladi, B.; Ghanoun, S.: EZ-Wave. Github. [online], 2016 [cit. 2018-03-18]. Dostupné z: <https://github.com/cureHsu/EZ-Wave>
- [19] Rose, A.; Ramsey, B.: Picking Bluetooth Low Energy Locks from a Quarter Mile Away. In *DEF CON, Las Vegas, NV, USA*, 2016 [cit. 2018-03-18].
- [20] Seri, B.; Vishnepolsky, G.: BlueBorne. Armis. [online], 2017 [cit. 2018-03-18]. Dostupné z: <http://go.armis.com/hubfs/BlueBorne%20Technical%20White%20Paper-1.pdf>
- [21] TechTarget: Do you need an IDS or IPS, or both? TechTarget. [online], 2009 [cit. 2018-03-19]. Dostupné z: <http://searchsecurity.techtarget.com/Do-you-need-an-IDS-or-IPS-or-both>



- [22] CZ.NIC: PaKon - sledování sítě (Parental Control). CZ.NIC. [online], 2018 [cit. 2018-03-20]. Dostupné z: <https://doc.turris.cz/doc/cs/howto/pakon>
- [23] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, 2016, doi:10.1109/CNSM.2016.7818417.



## Seznam použitých zkratk

<b>IoT</b>	Internet of Things
<b>IP</b>	Internet Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>VPN</b>	Virtual Private Network
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>CoAP</b>	Constrained Application Protocol
<b>AMQP</b>	Advanced Message Queuing Protocol
<b>DDoS</b>	Distributed Denial Of Service
<b>MITM</b>	Man In The Middle
<b>M2M</b>	Machine To Machine
<b>TCP</b>	Transmission Control Protocol
<b>QoS</b>	Quality of Service
<b>TLS</b>	Transport Layer Security
<b>RESTful</b>	Representational State Transfer
<b>UDP</b>	User Datagram Protocol
<b>HTTP</b>	Hypertext Transfer Protocol
<b>URI</b>	Uniform resource identifier
<b>DTLS</b>	Datagram Transport Layer Security
<b>ISM</b>	Industrial, Scientific and Medical

**AES** Advanced Encryption Standard

**PIN** Personal Identification Number

**QR** Quick Response

**ECDH** Elliptic-curve Diffie–Hellman

**BLE** Bluetooth Low Energy

**SIG** Special Interest Group

**NFC** Near Field Communication

**IDS** Intrusion Detection System

**IPS** Intrusion Prevention System

**SCADA** Supervisory Control And Data Acquisition

**NEMEA** Network Measurements Analysis

**UniRec** Unified Record

**RAII** Resource Acquisition Is Initialization

**RTT** Round-Trip Time

## **Instalační příručka**



## Obsah přiloženého CD

	readme.txt.....	stručný popis obsahu CD
	exe .....	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis .....	zdrojová forma práce ve formátu L <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	thesis.ps .....	text práce ve formátu PS