VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SNMP AGENT PRO IOT ZAŘÍZENÍ

SNMP AGENT FOR IOT DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **KATERYNA POLISHCHUK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. PETR MATOUŠEK, Ph.D., M.A.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Studentka: **Polishchuk Kateryna**
Program: Informační technologie
Název: **SNMP agent pro IoT zařízení**
SNMP Agent for IoT Devices
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se systémem pro monitorování síťových zařízení SNMP.
2. Prostudujte nástroje a knihovny pro tvorbu SNMP agenta, např. Net-SNMP, Agentuino, Agent++, které jsou vhodné pro monitorování IoT zařízení.
3. Navrhněte a implementujte komunikační část, která bude generická pro různá IoT zařízení.
4. Pro vybrané IoT zařízení zvolte vhodné MIB objekty pro monitorování. Popište jejich syntax jazykem ASN.1 a implementujte tyto objekty do SNMP agenta.
5. Otestujte funkčnost vytvořeného SNMP agenta. Na vybraných scénářích ukažte přínos monitorování IoT zařízení pomocí SNMP.
6. Zhodnoťte výsledek práce a navrhněte další rozšíření.

Literatura:

- William Stallings. 1998. *Snmp,Snmpv2,Snmpv3,And RMON 1 and 2* (3rd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA
- Savić, Mihajlo. (2016). Bridging the SNMP gap: Simple network monitoring the internet of things. Facta universitatis - series: Electronics and Energetics. 29. 475-487. 10.2298/FUEE1603475S.
- Hubin Deng, Guiyuan Liu, Lei Zhang. Analysis and Implementation of Embedded SNMP Agent. 4thConference on Computer and Computing Technologies in Agriculture (CCTA), Oct 2010, Nanchang,China. pp.96-102
- Net-SNMP Tutorial, Dostupné na http://www.net-snmp.org/tutorial/tutorial-5/toolkit/, červen 2019.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Matoušek Petr, Ing., Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 22. října 2019

Abstrakt

Cílem práce je vytvořit SNMP agenta pro monitorování IoT zařízení. Jelikož většina IoT zařízení nepodporuje protokol SNMP, chceme implementovat SNMP agenta, který bude shromažďovat informace o zařízeních a zapisovat shromážděná data na hodnoty proměnných v databázi MIB. Postup bude demonstrován na zařízeních D-Link, a právě dveřním senzoru a signálním zařízení, které komunikují s adaptérem přes rádiový signál Z-Wave.

Abstract

The aim of this work is to create a SNMP agent for monitoring IoT devices. Because most IoT devices do not support SNMP, we want to implement a SNMP agent that will collect device information and write the collected data to the values of the variables in the MIB. The procedure will be demonstrated on D-Link devices, as a door sensor and signaling device that communicate with the adapter via the Z-Wave radio signal.

Klíčová slova

Internet věcí, monitorování sítě, protokol SNMP, SNMP agent

Keywords

Internet of Things, System Network Management Protocol, Network monitoring, SNMP agent

Citace

POLISHCHUK, Kateryna. *SNMP agent pro IoT zařízení.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr Matoušek, Ph.D., M.A.

SNMP agent pro IoT zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Petra Matouška, Ph.D., M.A. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Kateryna Polishchuk
26. května 2020

Poděkování

Chtěla bych poděkovat panu Ing. Petru Matouškovi, Ph.D., M.A., který byl vždy ochotný poskytnout odbornou pomoc při vypracování práce.

Obsah

Kapitola 1

Úvod

Téma *Internet věcí* se stává každý rok populárnějším. IoT zařízení slouží pro automatizaci a bezpečnost domácnosti, zejména pro řízení a správu obytných a komerčních budov. Internet věcí umožňuje zařízením, aby byla vzdáleně kontrolována pomocí existující infrastruktury. Většina IoT zařízení komunikuje s kontrolérem pomocí bezdrátové komunikace typu ZigBee, Bluetooth, Z-Wave apod. Ukládání a zpracování dat o IoT zařízeních neprobíhá na samotném zařízení, ale v cloudu. Používání cloudu je důležité pro agregaci a analýzu dat, což může při velkém množství dat výrazným způsobem zlepšit výpočetní výkon. Senzory a zařízení shromažďují data a provádějí akce, samotné zpracování a analýza obvykle probíhá v cloudu. Přístup do cloudu umožňují mobilní aplikace, které poskytují jednotliví výrobci IoT zařízení. Uchování dat na cloud serverech má svoje nevýhody, například úniky a poškození dat, útoky na servery. Ke zpracování datových proudů slouží cloudová brána, která filtruje a případně blokuje data před provedením analýzy na cloudu, což způsobuje riziko ztráty dat. Data přenášena v rámci protokolů nemusí být zabezpečená. Útočník může napadnout síť a vytvořit spojení mezi zařízením a bránou nebo mezi bránou a serverem. Další nevýhodou této technologie je nemožnost získat informace o stavu IoT sítě mimo data z cloudu. Vhodným řešením je proto zabezpečit bezdrátovou komunikaci monitorovacím systémem, který umožní sběr dat ze zařízení lokálně.

Pro správu a monitorování síťových zařízení se používá systém SNMP, který umožňuje průběžný sběr nejrůznějších dat pro potřeby správy sítě a jejich následné vyhodnocování. IoT zařízení většinou nepodporují protokol SNMP, neboť nepoužívají TCP/IP, ale komunikují přes rádiový signál, například Z-Wave. Tato technologie využívá nízkoenergetické a vysokofrekvenční moduly, které jsou zabudovány do spotřební elektroniky, jako jsou svítidla, topná zařízení, zařízení pro kontrolu přístupu, domácí spotřebiče atd. Pro zajištění aktuální informace o stavu Z-Wave zařízení je vhodné rozšířit Z-Wave komunikaci o podporu systému SNMP. Tato podpora může pomoci organizacím používajícím IoT zařízení předejít nežádoucímu ohrožení IoT sítě.

Cílem této bakalářské práce je zabezpečit monitorování zařízení komunikujících přes protokol Z-Wave systémem SNMP, aby je bylo možné sledovat vzdáleně z řídicí stanice SNMP. Monitorovaná zařízení je potřeba vybavit agentem SNMP, který bude shromažďovat informace o zařízeních a ukládat tyto hodnoty do databáze.

V první kapitole budou podrobněji popsané základní koncepty správy sítě SNMP, architektura správy sítě a klíčové prvky, které SNMP tvoří. V další kapitole se podíváme na existující nástroje pro tvorbu agenta, zmíníme požadavky na hardware a software, výhody a nevýhody těchto nástrojů.

V třetí kapitole ukážeme, jak vypadá návrh architektury systému tvořící zabezpečení domácnosti, přenos dat mezi prvky systému a možnosti implementace.

Nakonec bude popsáno, jak implementovat agenta a databázi uchovávající data o zařízeních pomocí zvoleného nástroje. Dále bude popsáno, jak probíhá sběr dat z IoT komunikace, a budou popsány různé scénáře komunikace Z-Wave. V poslední kapitole otestujeme funkčnost vytvořeného agenta, ukážeme přínos monitorovaní IoT zařízení pomocí SNMP.

Kapitola 2

Systém SNMP

Termín Simple Network Manager protokol se používá k označení souhrnu specifikací pro správu sítě, které zahrnují samotný protokol SNMP, definici datových struktur a související koncepty. V této kapitole jsou vysvětleny základní koncepty správy sítě SNMP, architektura správy sítě, klíčové komponenty sítě SNMP a syntax objektů [10].

2.1 Architektura správy sítě

Model správy sítě, který se používá pro správu sítě TCP/IP, zahrnuje následující klíčové prvky:

- monitorovací systém – řídicí stanice (Network Management System, SNMP server) a SNMP agent,
- monitorované objekty definované pomocí jazyka SMI [8],
- uspořádání objektů do skupin, decentralizovaná správa objektů – databáze objektů MIB (Management Information Base) a
- komunikační protokol SNMP (SNMPv1, SNMPv2, SNMPv3).

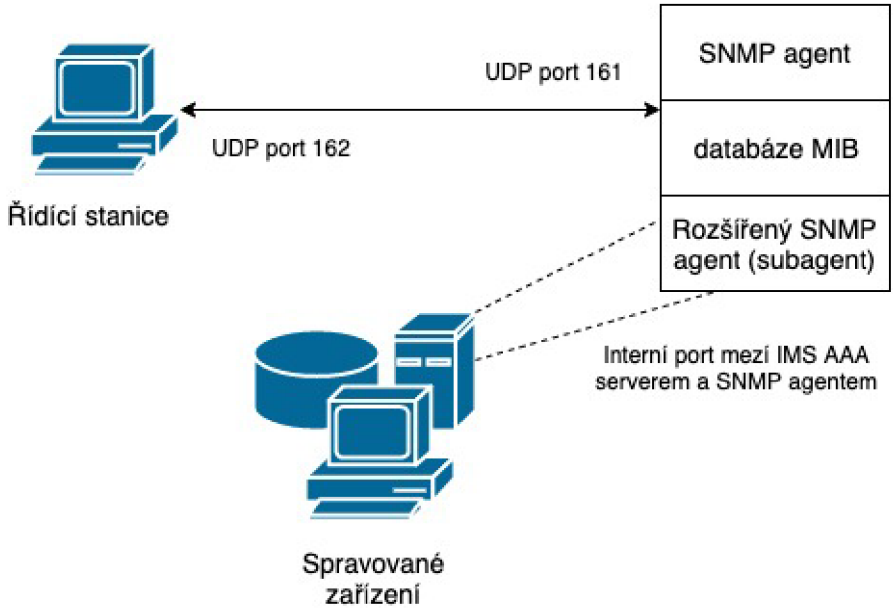
řídicí stanice je obvykle samostatné zařízení nebo vlastnost implementovaná do sdíleného systému, která slouží jako rozhraní pro správce sítě. řídicí stanici lze nainstalovat buď přímo na vyhrazeném serveru a spravovat na místě, nebo k ní přistupovat jako ke službě. Minimálně musí řídicí stanice obsahovat další vlastnosti:

- sadu aplikací pro analýzu dat, zotavení po poruše atd.,
- rozhraní, pomocí kterého může správce sítě monitorovat a řídit síť,
- schopnost převést požadavky správce sítě do skutečného monitorování a řízení vzdálených prvků v síti,
- databáze informací získaných z MIB všech spravovaných entit v síti [10].

Dalším aktivním prvkem v systému správy sítě je *agent*. Monitorovaná zařízení, jako jsou hosty, směrovače a rozbočovače, mohou být vybaveny agenty SNMP, aby je bylo možné spravovat z řídicí stanice. Agent odpovídá na požadavky řídicí stanice, kdy zasílá vyžádané

objekty obsahující monitorované údaje. Agent může také sám posílat asynchronní zprávy typu `trap`.

Na obrázku č. 2.1 je vidět schéma interakce agenta SNMP a manažera SNMP. Správce SNMP odesílá požadavky agentovi na portu UDP/161. Poté agent přijme paket a zpracuje jej. Během zpracování je generována odpověď, která je odeslána na port převzatý z původního požadavku. Agent SNMP tak poskytuje přístup SNMP manažerům k datům uloženým v databázi objektů.



Obrázek 2.1: Interakce SNMP manažera a SNMP agenta

Objekt je datovou proměnnou, která představuje jeden aspekt spravovaného agenta. Kolekce objektů je označována jako *management information base (MIB)* [10]. Agenti shromažďují informace o zařízeních a zapisují shromážděná data na hodnoty proměnných v databázi MIB. Řídicí stanice provádí monitorovací funkci získáním hodnoty objektů MIB. Řídicí stanice může způsobit, že se akce provede u agenta, nebo může změnit nastavení konfigurace u agenta úpravou hodnoty konkrétní proměnné.

Řídicí stanice a agenti komunikují protokolem pro správu sítě *SNMP*. Protokol Simple Network Management Protocol obsahuje následující klíčové funkce:

- načtení hodnoty objektů u agenta (*Get, GetNext, GetBulk*),
- nastavení hodnoty objektů u agenta (*Set*),
- agent musí umět upozornit řídicí stanici na významné události (*Trap*).

Komunikace mezi entitami protokolu se provádí výměnou zpráv, přenášených v rámci jednoho UDP datagramu pomocí základních pravidel kódování ASN.1. Zpráva se skládá z verze identifikátoru, názvu komunity SNMP a datové jednotky protokolu (PDU) [3].

2.2 Definice monitorovaných objektů

Standard RFC 1155 [9] popisuje strukturu a identifikaci monitorovaných dat (objektů) vyjádřených pomocí jazyka *SMI (Structure Management Information)*. Jazyk SMI definuje

pravidla, jak vytvářet objekty SNMP popsané notací ASN.1 (Abstract Syntax Notation). Každý objekt popisovaný pomocí ASN.1 má své *jméno, syntax* a *kódování* [7].

- *Jméno* objektu je dáno jednoznačným identifikátorem OID.
- *Syntax* objektu definuje abstraktní datový typ spojený s objektem, například typ INTEGER, OCTET STRING (základní datové typy ASN.1). Důležitým datovým typem je OBJECT IDENTIFIER, který obsahuje identifikátor OID pro daný objekt.
- *Kódování* určuje, jak jsou instance objektů reprezentovány při přenosu po síti. SMI používá pro kódování standard BER (Basic Encoding Rules) [6].

Základní datové typy ASN.1 použité pro tvorbu informací o monitorovaných objektech jsou uvedeny v tabulce 2.1. Na základě těchto datových typů si můžeme vytvořit vlastní objekt s položkami, které chceme sledovat.

Datový typ	Popis
INTEGER	32-bitové celé číslo definované ASN.1
OCTET STRING	binární či textový řetězec (až 64kB) definovaný ASN.1
OBJECT IDENTIFIER	přiřazeno ASN.1
Integer32	32-bitové celé číslo
Unsigned32	kladné 32-bitové celé číslo
IPaddress	32-bitová IP adresa, síťový formát
NetworkAddress	pro reprezentaci jiných typů adres
Counter32	32-bitový čítač, po přetečení se nastaví na 0
Counter64	64-bitový čítač
Gauge32	32-bitový čítač, po přetečení uchová max. hodnotu až do resetu
TimeTicks	čas měřený v setinách sekund od dané události
Opaque	neinterpretovaný řetězec podle ASN.1

Tabulka 2.1: Základní datové typy [7]

2.3 Uspořádání objektů v databázi MIB

Každé síťové zařízení přístupné protokolem SNMP obsahuje objekty definované v jedné či více MIB databázích, které představují hierarchicky uspořádanou sadu informací dostupných pro dané zařízení.

Standard RFC 2578 [8] definuje jazyk SMI, tzn. jakým způsobem bude prováděna manipulace s objekty a dále to, jak k těmto objektům budeme přistupovat. Přístup k objektům databáze je řešen pomocí objektového identifikátoru OID. Ten může mít textovou či číselnou formu [11].

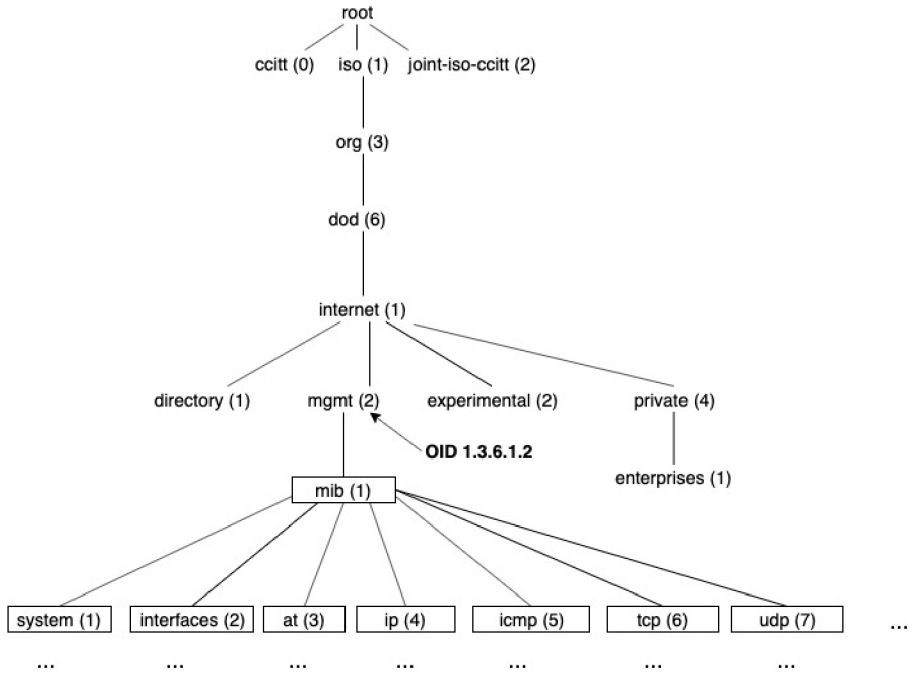
Struktura databáze MIB se skládá z objektů kořen, větev a list. Každý z těchto objektů nese dvojí označení, a to textové a číslicové. Každý jednotlivý výrobce si může zažádat o registraci vlastní větve. V této větvi si mohou výrobci daného zařízení vytvářet svou vlastní strukturu. Tyto nově vytvářené větve výrobců jsou zařazeny pod OID začínající 1.3.6.1.4 [11]. Deset základních skupin objektů v MIB-II je uvedeno v tabulce 2.2.

Každá skupina obsahuje množinu objektů, které obsahují další proměnné. Například skupina *system* obsahuje objekty *sysDescr, sysUpTime, sysObjectID, sysContact, sysName.*

Jméno podstromu	*OID*	*Popis*
system	1.3.6.1.2.1.1	operační systém: jméno OS, systémový čas, kontakt
interface	1.3.6.1.2.1.2	stav síťového rozhraní
at	1.3.6.1.2.1.3	překlad adres (address translation) - nepoužívá se
ip	1.3.6.1.2.1.4	IP adresa, směrovací informace
icmp	1.3.6.1.2.1.5	sleduje chyby ICMP
tcp	1.3.6.1.2.1.6	stav spojení TCP (closed, listen, synSent)
udp	1.3.6.1.2.1.7	statistiky UDP - přišlé/odešlé pakety
egp	1.3.6.1.2.1.8	statistiky EGP
transmission	1.3.6.1.2.1.10	objekty závisející na přenosovém médiu
snmp	1.3.6.1.2.1.11	počet vyslaných/přijatých SNMP paketů

Tabulka 2.2: Skupiny objektů definovaných v MIB-2 [7]

Obrázek č. 2.2 zobrazuje příklad stromové struktury objektů MIB.



Obrázek 2.2: Příklad stromové struktury MIB objektů

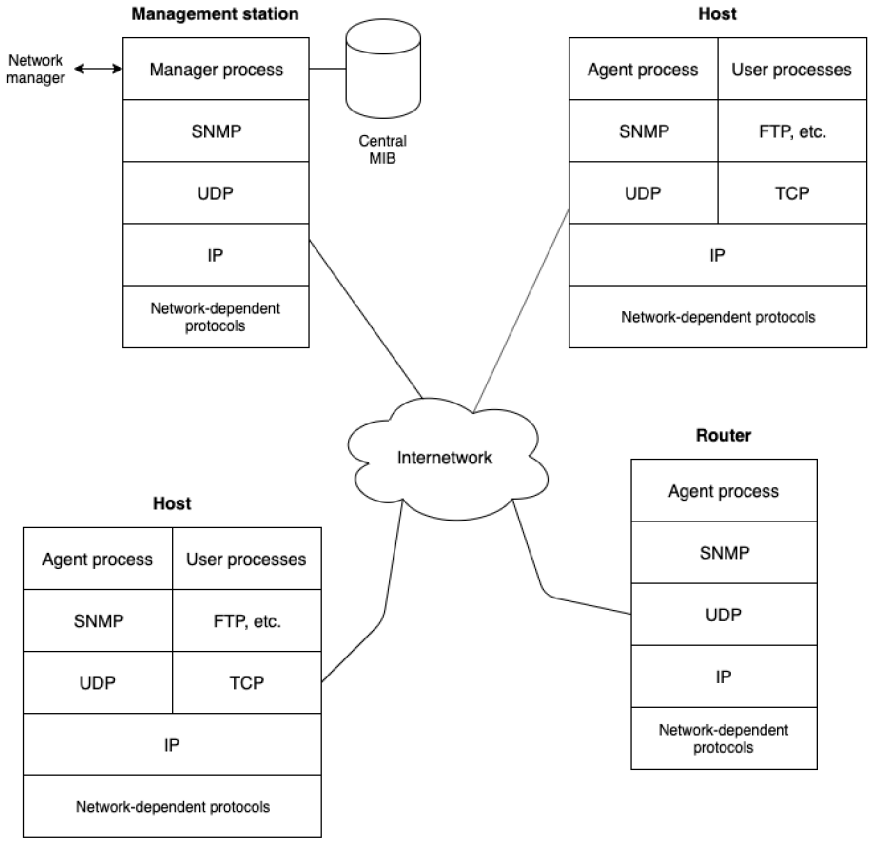
2.4 Protokol SNMP

Správa sítí vytvořených nad protokolovým profilem TCP/IP byla z počátku založená pouze na protokolu ICMP. S rostoucím počtem uzlů v síti však nebylo možné kontrolovat stav zařízení jednoduchými dotazy ICMP a začalo se uvažovat o návrhu systému pro správu sítí. Prvním byl protokol SGMP (Simple Gateway Motinoring Protocol) vydaný roku 1987 [4]. Z něho se po kratké době (v roce 1988) vyvinul protokol SNMP (Simple Network Manager Protocol) [7].

Protokol SNMP byl navržen jako aplikační protokol, tzn. SNMP pracuje na 7. úrovni modelu OSI. Je určen pro práci s protokolem User Datagram Protocol (UDP) [10]. Obrázek č. 2.3 ukazuje typickou konfiguraci protokolů pro SNMP.

Komunikační protokol SNMP je protokol typu dotaz-odpověď (request-response). Protože pracuje nad UDP, jedná se o nestavový protokol, tzn. každá výměna dat typu dotaz-odpověď je nezávislá [10].

Obvyklá činnost systému SNMP je taková, že řídicí stanice v pravidelných intervalech (např. 5 minut) sbírá provozní informace z monitorovaných zařízení, vyhodnocuje je a ukládá do databáze. Agent se tedy po většinu času chová jako pasivní účastník komunikace, který pouze čeká na přicházející žádosti. Jedná se tedy o komunikaci typu vyzývaní (polling). Teprve při výskytu mimořádných událostí posílá agent aktivní zprávy typu trap [7].



Obrázek 2.3: Konfigurace SNMP [10]

2.5 Shrnutí

V této kapitole jsme si představili architekturu systému pro správu sítě SNMP, jeho základní prvky a také to, jak mezi sebou tyto prvky komunikují. Podívali jsme se na definici objektů popsaných pomocí jazyka ASN.1, tzn. z čeho se skládá každý objekt.

Pak jsme se podívali, jak jsou objekty uspořádané v databázi MIB, jak přistupovat k těmto objektům a jak vypadá struktura databáze MIB.

Zjistili jsme, na jaké vrstvě pracuje protokol SNMP a podívali jsme se na typickou konfiguraci protokolu SNMP.

Systém SNMP vytváří strukturu agentu a řídicí stanici NMS, které sledují stav sítě. řídicí stanice a agenti komunikují pomocí protokolu SNMP, který pracuje na aplikační vrstvě. Všechna monitorovaná data jsou uložená v databázi MIB, ve které každý objekt má svoje jméno dané jednoznačným identifikátorem OID.

V další kapitole se podíváme na nástroje a knihovny pro tvorbu SNMP agenta, jejich klíčové vlastnosti, a na to, jaké jsou výhody a nevýhody těchto nástrojů.

Kapitola 3

Nástroje a knihovny pro tvorbu SNMP agenta

Následující kapitola popisuje nástroje a knihovny pro vytváření vlastního agenta SNMP, vlastnosti každého nástroje, výhody a nevýhody, závislost na operačním systému a také požadavky na hardware. Jelikož úkolem této bakalářské práce je implementovat SNMP agenta pro monitorování zařízení, které nepodporují prokol SNMP, je nutné zvolit si vhodný nástroj pro implementaci agenta.

3.1 Net-SNMP

Net-SNMP[1] je sada programů pro správu sítě používajících SNMP verzi v1, v2c a v3 včetně protokolu AgentX, transportní protokoly IPv4, IPv6, IPX, AAL5 a unixové sokety. Výhodou nástroje Net-SNMP je nezávislost na operačním systému. Net-SNMP je k dispozici pro mnoho unixových operačních systémů a také pro Microsoft Windows.

Net-SNMP obsahuje:

- Nástroje příkazového řádku pro:
  - načtení informace ze zařízení podporujícího SNMP buď pomocí jednoduchých požadavků (`snmpget`, `snmpgetnext`) nebo více požadavků (`snmpwalk`, `snmptable`, `snmpdelta`),
  - manipulace s konfiguračními informacemi na zařízeních podporujících SNMP (`snmpset`),
  - načtení informací ze zařízení podporujících SNMP (`snmpdf`, `snmpnetstat`, `snmpstatus`),
  - převod mezi numerickými a textovými formami MIB OID a zobrazování obsahu a struktury MIB (`snmptranslate`).
- Grafický prohlížeč MIB (`tkmib`)
- Démonovou aplikaci pro příjem oznámení SNMP (`snmptrapd`). Vybraná oznámení lze logovat (do syslogu, do protokolu událostí NT nebo do jednoduchého textového souboru), přeposílat do jiného systému správy SNMP nebo předávat externí aplikaci.

[1] Viz http://www.net-snmp.org/

- Rozšířitelného agenta pro odpovědi na dotazy SNMP na informace o správě (`snmpd`). To zahrnuje vestavěnou podporu pro širokou škálu informačních modulů MIB a lze ji rozšířit pomocí dynamicky načtených modulů, externích skriptů, příkazů a protokolů SNMP multiplexing (SMUX) a Agent Extensibility (AgentX).
- Knihovnu pro vývoj nových aplikací SNMP s rozhraním C i perl API.

3.2 Agentuino

Agentuino[2] je jednoduchá knihovna SNMP pro platformy Arduino podporující verzi 1. Software podporuje:

- Typy PDU
- Get-Request
- Set-Request
- Response

Agentuino také podporuje datové typy: Null, Boolean, Bits, Octet-String, Object-Identifier, Integer and Integer32, Counter and Counter64, IP adresy a další.

3.2.1 Omezení

Agentuino závisí na množství dostupného SRAM. Maximální délka Community Name (Get/Set): *20 bajtů*. Maximální délka identifikátoru objektu: *64 bajtů*. Maximální Value Length: *64 bajtů*. Maximální délka paketu: *153 bajtů* [2].

3.2.2 Požadavky na hardware [2]:

- Procesorová deska Arduino
- SRAM 2k nebo vyšší
- Paměť Flash 30k nebo vyšší
- Arduino Ethernet Shield
- Čip WizNet nebo čip podporující UDP a standardní knihovnu Ethernet

3.2.3 Požadavky na software [2]:

- Arduino IDE 0019 nebo vyšší[3]
- Ethernetová knihovna (výchozí s 0019 nebo vyšší)
- Knihovna SPI (výchozí hodnota 0019 nebo vyšší)
- Knihovna Streaming[4]

[2] Viz https://github.com/1sw/Agentuino
[3] Viz http://arduino.cc/en/Main/Software
[4] Viz http://arduiniana.org/library/streaming/

- Knihovna Flash[5]
- Knihovna MemoryFree[6]
- Knihovna Agentuino

3.3 AgentPP

Sada nástrojů AgentPP[7] slouží pro vývoj agentů, aplikací SNMP (Java/C++) a také MIB databází. Obsahuje nástroj AgentPro[8], Java framework SNMP4J[9], C++ framework Agent++[10] a další nástroje.

Agent++ implementuje kompletní modul protokolu a expediční tabulku pro vývoj SNMP agentů. Podporuje všechny verze SNMP a poskytuje View based Access Control (VACM). Agent++ podporuje Linux, Solaris, HPUX, AIX, Windows 7/8/XP a může být použit s jinými platformami a mnoha vestavěnými operačními systémy.

AgenPro generuje Java a C++ SNMP agenty ze sady MIB. Obsahuje back-end generování otevřeného zdrojového kódu, který dává uživateli plnou kontrolu nad formátem, rozsahem, cílovým API a dokonce i programovacím jazykem generovaného kódu. Vestavěný simulátor agentů SNMPv1/2c/3 poskytuje okamžitě spustitelného agenta [1].

Klíčové vlastnosti AgentPro jsou [1]:

- Back-end pro generování zdrojového kódu na základě šablon. Šablony pro generování kódu jsou zahrnuty pro Agent++/AgentX++ a také pro SNMP4J.
- Editor MIB se zvýrazněním syntaxe, kompilátorem pozadí a pěkným tiskem.
- Formátování souborů MIB a export HTML modulů MIB do HTML.
- Inicializace hodnoty objektu SNMP a generování kódu šablony trap/oznámení (již integrované v šablonách SNMP4J).

3.4 Shrnutí

V dnešní době se používají ke tvorbě agenta SNMP nástroje Net-SNMP, Agentuino a AgentPP.

Na rozdíl od knihovny Agentuino, která je určená pro platformy Arduino, Net-SNMP a AgentPP jsou k dispozici pro mnoho unixových operačních systémů a také pro Microsoft Windows. Knihovna Agentuino má mnohem více požadavků na hardware a software, nemá grafický prohlížeč MIB na rozdíl od Net-SNMP nebo editor MIB, který má software AgentPP.

Net-SNMP je nástroj pro vývoj nových aplikací SNMP v jazyku C, který podporuje rozšíření hlavního agenta za použití dynamicky načtených modulů, externích shell a Perl skriptů a protokolu AgentX. AgentPP umožňuje tvorbu agenta v jazyku Java nebo C++.

[5] Viz http://arduiniana.org/libraries/flash/
[6] Viz http://www.arduino.cc/playground/Code/AvailableMemory
[7] Viz https://agentpp.com/
[8] Víz https://agentpp.com/tools/agenpro.html
[9] Víz https://agentpp.com/api/java/snmp4j.html
[10] Víz https://agentpp.com/api/cpp/snmp_pp.html

Pro implementaci agenta jsem zvolila nástroj Net-SNMP. Konfigurace systému SNMP je popsána v příloze A.

V další kapitole bude popsán návrh aplikace, diagram přenosu dat mezi jednotlivými prvky systému a možnosti implementace.

Kapitola 4

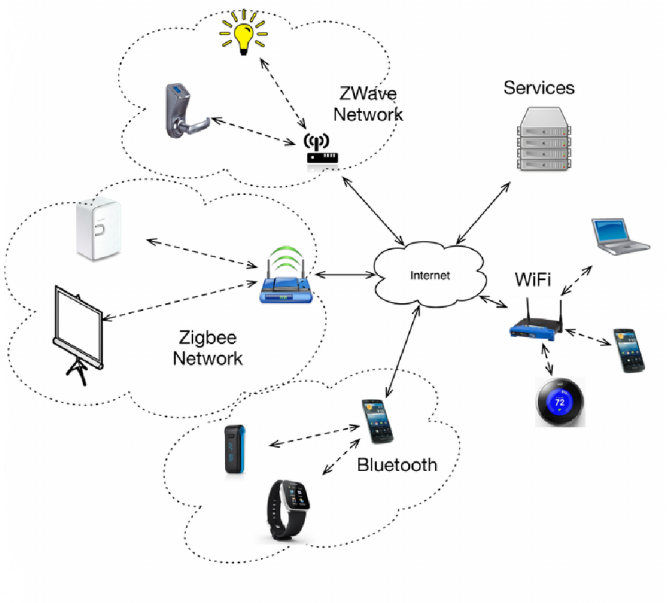
Návrh monitorovacího systému

Jeden z požadavků na navrhovaný systém je definice základních prvků, které budou systém tvořit. Pro demonstraci funkčnosti aplikace budeme používat testovací sadu, která obsahuje dveřní senzor a signalizační zařízení. Jedná se o přístroje výrobce D-Link ze sady inteligentní domácí bezpečnosti[1].

V této kapitole budeme psát podrobněji o zařízeních, struktuře IoT sítě a o tom, jak bude probíhat přenos dat mezi jednotlivými prvky systému. Dále definujeme požadavky na systém a podíváme se na možnosti implementace systému.

4.1 Struktura IoT sítě

Na obrázku č. 4.1 je zobrazená obecná struktura bezdrátové komunikace. Protokol Z-Wave



Obrázek 4.1: Struktura IoT sítě [zdroj]

a Zigbee využívají například termostaty, senzory pohybu, alarmy, osvětlení, klimatizace, IP

[1] Viz https://eu.dlink.com/uk/en/products/dch-107kt-smart-home-security-kit

kamery, zámky na dveře, ovladače audio/video techniky atd. Technologie Z-Wave a Zigbee používají smíšenou topologii (mesh), mají nízký výkon a jsou navržené pro přenos malého množství dat na krátké až střední vzdálenosti. Z-Wave využívá síťové pásmo menší než 1 GHz (frekvence se liší v různých zemích), což umožňuje bezkonfliktní provoz při souběhu s Wi-Fi a dalšími systémy založenými na standardu IEEE 802.11. Technologii Bluetooth využívají mobilní telefony, tablety, hodinky, sluchátka, počítače, USB atd. Bluetooth nepodporuje žádný vzdálený přístup, jen point-to-point. Nevýhodou této technologie je přenos dat na krátkou vzdálenost, maximálně do 10 metrů, protože se k přenosu dat používá velmi malý výkon. Technologie Wi-Fi souvisí s připojením počítačů, tabletů a mobilních telefonů k internetu. Výhodou Wi-Fi technologie je její snadná integrace, možnost připojení i mimo běžné pracovní prostředí v případě propojení s veřejnou sítí a snadné rozšiřování sítě při přidávání dalších klientů.

V současném ekosystému IoT lze různé komponenty IoT obecně rozdělit do tří tříd: uzly senzorů, uzly brány a služby IoT. Typické uzly senzorů sestávají z domácích spotřebičů nebo senzorů sledujících fyzické prostředí, které mají nízké výpočetní zdroje, přísná omezení energie a omezené komunikační zdroje. Uzel brány funguje jako agregátor dat senzorů a poskytuje připojení k jiným uzlům senzorů a poskytovatelům služeb. Služby IoT shromažďují data z různých uzlů brány, zpracovávají je v cloudu a poskytují služby specifické pro uživatele nebo události pomocí grafického rozhraní, oznámení nebo aplikace[2]. Chytré domácí mobilní aplikace pomáhají řešit problém správy více zařízení v inteligentních domácnostech z jednoho centrálního uživatelského ovládání. Uživatelé mohou například sledovat stav zařízení a upravovat ho podle svých požadavků vzdáleně. Nevýhodou technologie je nemožnost získat stav zařízení mimo data z cloudu, protože technologie nepodporuje jiný monitorovací systém.

4.2 Prvky systému

Sada pro zabezpečení domácnosti (Smart Home Security Kit) obsahuje rozbočovač[3], kameru[4], dveřní senzor[5] a signalizační zařízení, tj. sirénu[6].

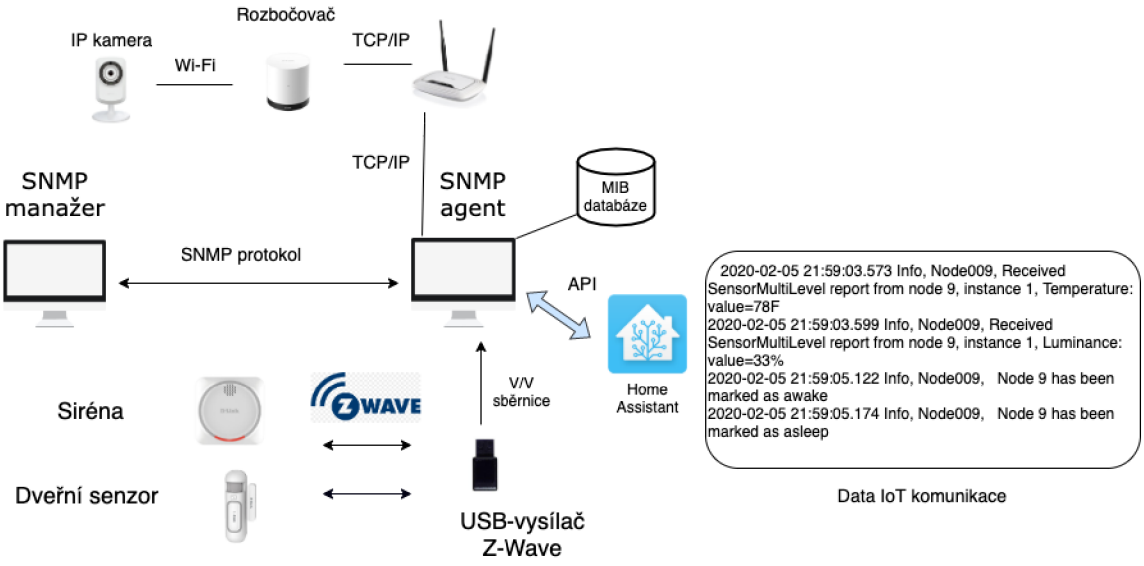
- *Rozbočovač* spojuje zařízení myDlink přes Wi-Fi a Z-Wave, aby spolu komunikovaly. Je nutné připojení k existující domácí Wi-Fi síti.
- *Kamera* se připojuje ke stávající Wi-Fi síti, tzn. přenáší pakety přes rozbočovač a zasílá upozornění při detekci pohybu nebo zvuku.
- *Dveřní senzor* komunikuje s rozbočovačem přes rádiový signál Z-Wave. Detekuje otevření dveří nebo okna a posílá upozornění.
- *Siréna* představuje inteligentní zvukové výstražné zařízení s šesti vestavěnými zvuky. Siréna nepodporuje IP protokol a komunikuje přes rádiový signál Z-Wave.

Na obrázku č. 4.2 je vidět, jak vypadá navrhovaná topologie prvků. Pro demonstraci funkčnosti aplikace budeme používat jen ty prvky, které které nepoužívají TCP/IP, ale komunikují přes rádiový signál Z-Wave, tzn. dveřní senzor a sirénu. Data o zařízeních se obvykle

[2] Viz Semantic Gateway as a Service Architecture for IoT Interoperability
[3] Viz https://eu.dlink.com/uk/en/products/dch-g020-mydlink-connected-home-hub
[4] Viz https://eu.dlink.com/cz/cs/products/dcs-935l-monitor-hd
[5] Viz https://eu.dlink.com/uk/en/products/dch-z110-mydlink-door-window-sensor
[6] Viz https://eu.dlink.com/uk/en/products/dch-z510-mydlink-home-siren

ukládají do cloudu, ale zde je chceme analyzovat lokálně. K tomu budeme potřebovat USB adaptér pro Z-Wave síť[7], který bude komunikovat se zařízeními. Zařízení implementuje virtuální rozhraní používané aplikací Z-Wave.

Daný postup vytvoření monitorovacího systému SNMP pro bezdrátové sítě lze aplikovat i na jiné IoT sítě, které lokálně používají rádiové spojení, například typu Bluetooth, Zigbee apod. Krátký postup vytvoření monitorovacího systému bude popsán v sekci 4.6.



Obrázek 4.2: Topologie prvků tvořící zabezpečení domácnosti

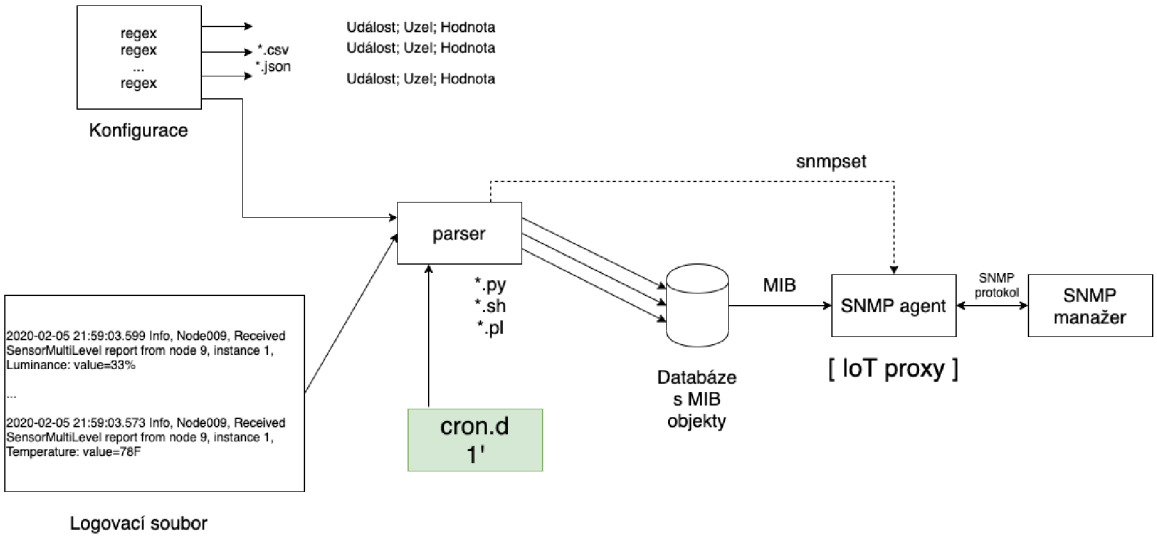
Navrhovaný systém obsahuje další prvek – *řídicí stanice NMS*, která monitoruje data o každém zařízení. Na počítači bude běžet *SNMP agent*, který bude shromažďovat informaci o zařízeních a zapisovat shromážděná data na hodnoty proměnných v databázi MIB. Informace o zařízeních podporujících Z-Wave bude možné získat z logovacího souboru aplikace Home Assistant[8]. Popis instalace aplikace Home Assistant je v příloze B.1.

4.3 Přenos dat mezi prvky systému

Na obrázku č. 4.3 je vidět, jak probíhá přenos dat mezi jednotlivými prvky systému. Logovací soubor nástroje Home Assistant obsahuje určité události, které zařízení posílají přes signál Z-Wave USB-vysílači. Každá událost musí obsahovat čas, uzel a příkaz (zprávu), který obsahuje určitou informaci o zařízení. Například je to informace o tom, jakou mají aktuální teplotu, svítivost, stav, typ notifikace v určitý okamžik času. Podrobněji bude formát přenášejících zpráv popsán v sekci 5.4.

Důležitou částí systému jsou pravidla pro parsování logovacího souboru. Konfigurační soubor bude obsahovat množinu pravidel pro vyhodnocování události. Každé události odpovídá regulární výraz, pomocí kterého jde událost jednoduše vyhledat v logovacím souboru.

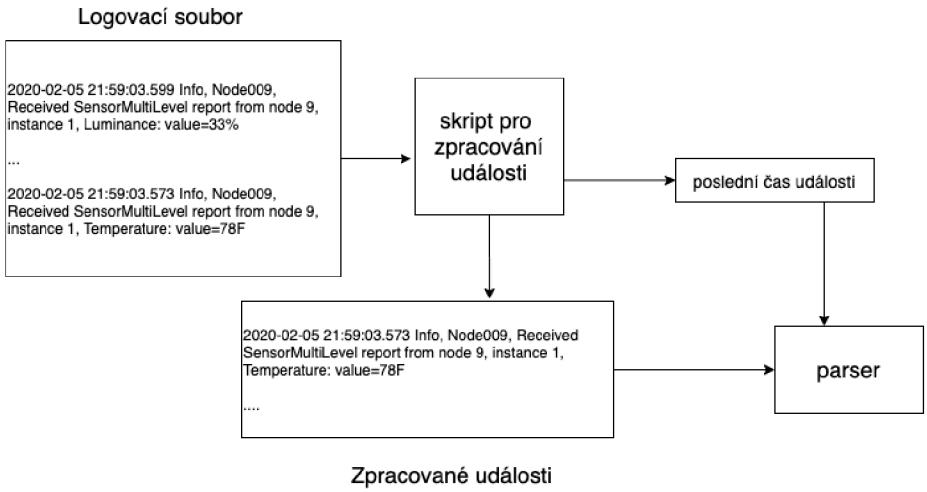
[7] Víz https://z-wave.me/download/ZMEXUZB1_Manual1.pdf
[8] Viz https://www.home-assistant.io/



Obrázek 4.3: Přenos dat mezi prvky systému

Regulární výraz je nejjednodušší řešení pro popis jazyka Z-Wave komunikace, jelikož logovací soubor je složitě interpretovatelný pro uživatele. Jiným možným řešením zpracovaní logovacího souboru je konečný automat.

Jedním z hlavních prvků systému je parsovací skript nebo jinak řečeno analyzátor, který prochází nejnovější události, vyhodnocuje události pomocí vytvořených syntaktických pravidel a získané hodnoty ukládá do proměnných v databázi MIB. Skript má sdílenou paměť, kam ukládá časovou značku poslední události, se kterou pak porovnává novější události. Pro optimalizaci a zajištění nejnovějších událostí navržené řešení obsahuje skript, který předem zpracovává logovací soubor z aplikace Home Assistant, vyhazuje starší události a připravuje nový soubor, který je vstupem pro parser. Zpracování události se bude skládat ze dvou částí, návrh je znázorněný na obrázku č. 4.4.



Obrázek 4.4: Zpracování logovacích informací

Parser nemusí běžet na stejném stroji s agentem, ale agent mu může poskytnout vzdálený přístup.

SNMP agent čte hodnoty objektů z MIB databáze a odpovídá na požadavky manažera vracením hodnot těchto objektů. Abychom znali aktuální hodnoty zařízení, je řešením implementovat parser do démonu `cron.d`. Tento démon spouští parsovací skript automaticky jednou za čas (například jednou za minutu), tím pádem parser ukládá do databáze nejnovější získanou hodnotu uzlu.

Výsledkem by měl být systém, který běží na pozadí a automaticky vyhodnocuje události o zařízeních.

4.4 Požadavky na systém

Systém vyžaduje:

- IoT zařízení a adaptér pro síť,
- podporu Python poslední verze,
- nástroje Home Assistant, tj. virtuálního prostředí `venv`, ve kterém nástroj bude běžet,
- knihovnu `python-openzwave`[9](pokud používáme Z-Wave síť),
- knihovnu Net-SNMP verzi 5.8 a
- Unixový operační systém.

4.5 Možnosti implementace

Jsou dva způsoby ukládání objektů do MIB databáze a získání hodnot objektů:

1. Parser ukládá hodnoty objektů do sdílené paměti (souboru), ze které bude agent hodnoty číst.
2. Parser volá funkci `snmpset`, která ukládá hodnoty objektů do MIB databáze.

Vhodným formátem uchování pravidel konfigurace je formát `csv` nebo `json`. Parser je možné implementovat pomocí skriptovacích jazyků Python, Perl nebo bash kvůli jednoduššímu zpracování regulárních výrazů a podpoře knihoven pro SNMP. Například Python podporuje knihovnu `pysnmp`[10]. Jazyk Perl a bash obsahují modul pro práci se SNMP v balíčku Net-SNMP[11].

Pokud SNMP agent bude běžet na různých strojích s parserem, je potřeba implementovat agenta s podporou proxy, aby k němu šlo přistupovat vzdáleně.

Hlavního SNMP agent je možné nakonfigurovat s podporou vytvořených objektů nebo ho rozšířit o subagent. Navržené řešení používá druhý způsob. Je to vlastní proces, který se připojuje k hlavnímu agentovi pomocí protokolu AgentX. Postup tvorby bude popsán v sekci 5.3.

Další kapitola obsahuje podrobný popis implementace každého modulu systému.

[9] Víz github repositář https://github.com/OpenZWave/python-openzwave
[10] Víz http://snmplabs.com/pysnmp/index.html
[11] Víz http://www.net-snmp.org/wiki/index.php/Tutorials#Coding_Tutorials

4.6 Návod k použití

Navržený systém umožňuje správu sítě a průběžný sběr nejrůznějších dat o IoT zařízeních mimo data z cloudu pomocí systému SNMP. Tato práce popisuje postup vytvoření SNMP agenta v jazyku C, MIB databázi a navrhuje analyzátor, který interpretuje data z logovacího souboru na hodnoty proměnných v databázi MIB. Daný postup lze aplikovat na IoT sítě, které komunikují bezdrátově a používají rádiové spojení, například Z-Wave, Bluetooth, Zigbee apod. Tato práce demonstruje funkčnost aplikace na zařízeních Z-Wave. Postup implementace systému SNMP pro jakékoliv bezdrátové sítě vypadá následovně:

1. Zvolit zařízení a adaptér pro konkrétní síť.
2. Nainstalovat nástroj pro zpracování události Home Assistant a knihovnu pro adaptér.
3. Popsat události, které se budou odchytávat pomocí regulárních výrazů.
4. Vytvořit agenta a definici objektů MIB.

Kapitola 5

Implementace

SNMP software, který zpracovává požadavky SNMP v síťovém uzlu, se nazývá agent. Agent Net-SNMP (snmpd) je zodpovědný za zpracování příchozích požadavků protokolu SNMP. V této kapitole je popsaná implementace rozšiřitelného agenta (subagenta), dále jakým způsobem agent komunikuje s manažerem a tvorba MIB databáze, tzn. definice typických uzlů a objektů, které dodržují určitou syntax.

5.1 Vytváření MIB databáze

Nejdůležitější části pro napsání vlastní MIB je seznam objektů, které budou uloženy v MIB. Tyto objekty musí striktně dodržovat syntax ASN.1. Databáze může obsahovat skalární hodnoty a tabulky. SNMP MIB má stromovou strukturu, kde kořen stromu a standardní objekty jsou definovány IETF. Pro vložení nových objektů MIB je nutné zařadit je do stromu MIB objektů.

Pokud chceme definovat objekty MIB, které chceme sdílet se světem, potřebujeme ve stromu místo, které nebude v konfliktu s jinými objekty. Proprietární větev stromu je určena organizacím (a jednotlivcům) pro definování jejich vlastních objektů. OID pro vlastní MIB přiděluje IANA na stránce Pen Application.[1]

Pokud definujeme objekty MIB, které budeme používat pouze interně, můžeme je vložit do experimentálního stromu. Jen si musíme dát pozor na překrývání OID.[2]

Každá MIB databáze obsahuje jako první řádek definice MIB, například:

```
IOT-MIB DEFINITIONS ::= BEGIN
```

Dále následují definice z jiných MIB:

```
IMPORTS
        netSnmpExamples         FROM NET-SNMP-EXAMPLES-MIB
OBJECT-TYPE, Integer32,
MODULE-IDENTITY         FROM SNMPv2-SMI
MODULE-COMPLIANCE, OBJECT-GROUP         FROM SNMPv2-CONF
DisplayString           FROM SNMPv2-TC;
```

[1] Viz http://pen.iana.org/pen/PenApplication.page

[2] Viz http://net-snmp.sourceforge.net/wiki/index.php/Writing_your_own_MIBs

Při napsání vlastní MIB jsem použila ukázkový příklad Net-SNMP a definovala vlastní MIB do větve *netSnmpExamples 4*. Zde je příklad definice *iotMIB* databáze včetně krátkého popisu databáze.

```
iotMIB MODULE-IDENTITY
    LAST-UPDATED "201912010000Z"            -- 22 November 2020, midnight
    ORGANIZATION "VUTBR"
    CONTACT-INFO "postal:   Kateryna Polishchuk
                  email:    xpolis03@fit.vutbr.cz"
    DESCRIPTION  "A mib for monitoring D-Link Corporation DCH-Z510 Siren."
    ::= { netSnmpExamples 4 }
```

Definice typických uzlů MIB vypadá následovně:

```
    iotMIBObjects      OBJECT IDENTIFIER ::= { iotMIB 1 }
    iotAgentModules    OBJECT IDENTIFIER ::= { iotMIBObjects 1 }
```

V tomto příkladu budou definované objekty *sirenState* a *sensorTemperature*. Definice dalších objektů bude popsaná v příloze D.

```
sirenState OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..3))
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION "Status of the device may take on values off/on"
    DEFVAL { "off" }
    ::= { iotAgentModules 2 }

sensorTemperature OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Door/Window temperature"
    DEFVAL { 0 }
    ::= { iotAgentModules 5 }

END -- konec definice
```

Jakmile je databáze napsaná, je potřeba přemístit soubor do konfigurační složky `/usr/local/share/snmp/mibs`, která obsahuje definici všech dostupných MIB.

5.2 Registrace objektů MIB

Přidání MIB do konfigurační složky ale neznamená, že agent bude automaticky vracet hodnoty z této MIB. Agent musí být explicitně rozšířen o podporu nových objektů MIB. Nejjednodušší metoda rozšíření agenta je převod definice MIB v ASN.1 do jazyka C, který lze zkompilovat do samotného agenta za účelem implementace požadovaného modulu.

Pro generování modulů existuje nástroj `mib2c`[3]. Nástroj je navržen tak, aby převzal část stromu MIB (jak je definováno v souboru MIB) a vygeneroval kostru kódu nezbytnou k implementaci. Příklad generování kódu `sirenState.c` a `sirenState.h`:

```
$ mib2c -c mib2c.scalar.conf sirenState
```

Podíváme se podrobněji na strukturu vygenerovaného kódu.
Funkce `netsnmp_create_handler_registration` vytvoří strukturu pro registrace s novým popisovačem (handler) MIB. Parametr `HANDLER_CAN_RWRITE` povoluje čtení a zápis hodnoty objektu, parametr `HANDLER_CAN_RONLY` jen zápis.

```
/*
* proměnné nutné pro registraci
*/
netsnmp_handler_registration *reginfo;
static netsnmp_watcher_info watcher_info;
int watcher_flags;

const oid sirenState_oid[] = { 1,3,6,1,4,1,8072,2,4,1,1,2 };

reginfo = netsnmp_create_handler_registration("open_connections_list",
NULL, sirenState_oid, OID_LENGTH(sirenState_oid), HANDLER_CAN_RWRITE);

/* Funkce vytváří informace o sledování řetězce objektu.*/
watcher_flags = WATCHER_MAX_SIZE;
netsnmp_init_watcher_info(&watcher_info, sirenState_value,
              strlen(sirenState_value), ASN_OCTET_STR, WATCHER_MAX_SIZE);

/* Funkce registruje proměnnou, zpřístupní ji a umožní její zápis.*/
netsnmp_register_watched_scalar(reginfo, &watcher_info);
```

Příklad generování MIB modulů pro objekty typu integer:

```
$ mib2c -c mib2c.int_watch.conf sensorTemperature
```

Nástroj vygeneruje kostru kódu, který má následující formát:

```
long sensorTemperature = 0; /* implicitní hodnota */

void init_sensorTemperature(void){
    netsnmp_handler_registration *reg;

    const oid sensorTemperature_oid[] = { 1,3,6,1,4,1,8072,2,4,1,1,5 };
    static netsnmp_watcher_info sensorTemperature_winfo;

    reg = netsnmp_create_handler_registration(
            "sensorTemperature", NULL,
```

[3] Viz http://www.net-snmp.org/docs/man/mib2c.html

```
            sensorTemperature_oid, OID_LENGTH(sensorTemperature_oid),
            HANDLER_CAN_RWRITE);
    netsnmp_init_watcher_info(&sensorTemperature_winfo, &sensorTemperature,
    sizeof(long), ASN_INTEGER, WATCHER_FIXED_SIZE);

    if (netsnmp_register_watched_scalar(reg,
    &sensorTemperature_winfo) < 0){
        snmp_log(LOG_ERR, "Failed to register watched sensorTemperature");
    }

}
```

5.3 Tvorba subagenta

Tato část popisuje implementace vlastního procesu, který se připojuje k hlavnímu agentovi pomocí protokolu AgentX. Jak bude řečeno v příloze A, pro podporu AgentX je nutné do souboru snmpd.conf vložit řádek `master agentx`.

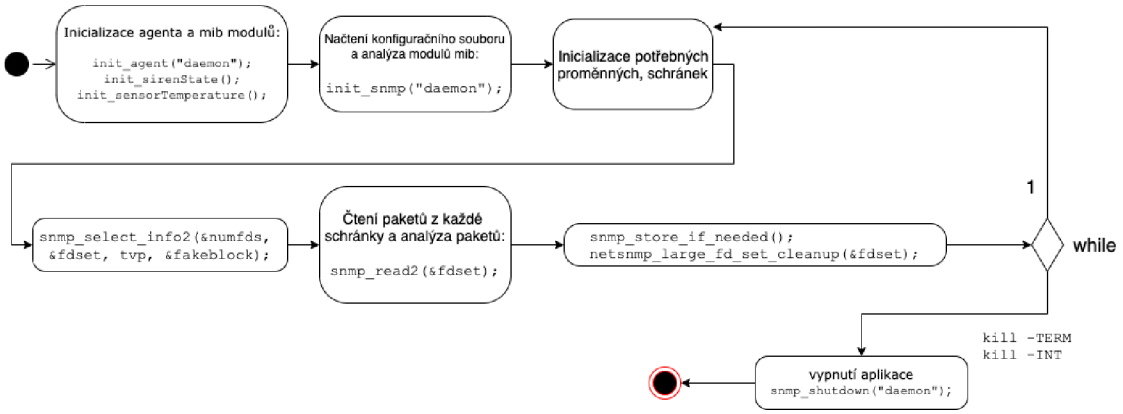
Pro napsání SNMP démona budeme používat balíčky *net-snmp/net-snmp-config.h, net-snmp/net-snmp-includes.h, net-snmp/agent/net-snmp-agent-includes.h*
Hlavní kód agenta vypadá následovně:

```
int main (int argc, char **argv) {
  int agentx_subagent=1; /* 1 - pokud chceme být SNMP master agent */
  int background = 0; /* 1 - pokud chceme běžet na pozadí */
  int syslog = 0; /* 1 - pokud chceme používat syslog */

  /* vypsat log chyby do syslogu nebo stderr */
  if (syslog)
    snmp_enable_calllog();
  else
    snmp_enable_stderrlog();

  /* agent je agentx subagent? */
  if (agentx_subagent) {
    /* nastaví agenta jako agentx client. */
    netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID,
                           NETSNMP_DS_AGENT_ROLE, 1);
  }

  /* běží na pozadí, pokud je vyžadováno */
  if (background && netsnmp_daemonize(1, !syslog))
      exit(1);

  /* inicializace tcpip, pokud je nutné */
  SOCK_STARTUP;
```

```
  /* inicializace knihovny pro agenta */
  init_agent("snmp-daemon");

  /* inicializace mib kodu objektu */
  init_sirenState();
  init_sensorTemperature();

  /* inicializace vacm/usm access control */
  if (!agentx_subagent) {
      init_vacm_vars();
      init_usmUser();
  }

  /* inicializace snmp demona */
  init_snmp("snmp-daemon");

  /* pokud chceme být snmp master agentem, musíme inicializovat porty */
  if (!agentx_subagent)
    /* otevřít port, na kterém budeme poslouchat (implicitní je udp:161) */
    init_master_agent();

  /* pro ukončení procesu */
  keep_running = 1;
  signal(SIGTERM, stop_server);
  signal(SIGINT, stop_server);

  snmp_log(LOG_INFO,"snmp-daemon is up and running.\n");

  /* tato funkce kontroluje pakety přicházející na port SNMP a
  zpracovává je (snmp_read); pokud jsou některé nalezeny, pomocí select().
  Pokud je argument nenulový, volaní funkci blokuje, dokud nedojde paket */
  while(keep_running) {
    snmp_run(1); /* 0 == neblokovat */
  }

  /* shutdown */
  snmp_shutdown("snmp-daemon");
  SOCK_CLEANUP;

  return 0;
}
```

Funkční diagram činnosti subagenta je na obrázku č. 5.1. Hlavním úkolem agenta je přijímat pakety přicházející na port SNMP 161 a zpracovávat je.

Nejdřív je potřeba inicializovat strukturu obsahující sadu deskriptorů souboru. Funkce `snmp_read()` přistupuje k množině schránek a čte z nich pakety, dále je analyzuje. Vý-



Obrázek 5.1: Diagram činnost SNMP agenta

sledná datová jednotka protokolu (PDU) je předána do rutiny zpětného volání pro relaci (snmp_session[4]). Pokud se zpětné volání úspěšně vrátí, PDU a její požadavek budou odstraněny.

5.4 Sběr dat z IoT komunikace

Data z IoT komunikace ukládá do logovacího souboru nástroj Home Assistant. Každý řádek souboru obsahuje časovou značku události, uzel (zařízení), pro který je vytvořená událost, typ události a samotnou zprávu. Níže je uveden příklad logovacího souboru:

```
2020-02-08 15:25:22.025 Detail, Node011, Received: 0x01, 0x09, 0x01, 0x41,
0xd3, 0x9c, 0x01, 0x04, 0x10, 0x05, 0xe9
```

Některé aplikační zprávy mají formát 16bitových hodnot a většinou je struktura následující [5]:

- 0x01 (1B, začátek zprávy)
- délka zprávy (1B)
- typ zprávy (1B, 0x00 - request, 0x01 - response)
- funkce (1B, 0x04, 0x13)
- ID uzlu (1B)
- délka dat (1B)
- typ příkazu (1B, např. 0x84 Wakeup atd.)

Každý uzel také obsahuje seznam příkazů, které reprezentují atributy každého zařízení, například:

- COMMAND_CLASS_BASIC
- COMMAND_CLASS_SWITCH_BINARY

[4] Viz http://net-snmp.sourceforge.net/dev/agent/structsnmp__session.html

- COMMAND_CLASS_SENSOR_BINARY atd[5].

Nevýhodou nástroje Home Assistant je nečitelnost a složitá interpretace příkazu z logovacího souboru. Proto je potřeba implementovat formální pravidla, pomocí kterých se události vyhodnotí a uloží do MIB databáze v čitelném formátu. Úkolem této části je vybrat vhodné objekty, které chceme u zařízení monitorovat, definovat formát příkazů a vytvořit konfigurační soubor, který bude obsahovat regulární výrazy (dále jen RV) pro získání hodnot objektů.

Navržené řešení obsahuje jednoduché události, které vypovídají o stavu zařízení a také o teplotě a svítivosti senzoru. Události mají následující formát zpráv:

```
Received SensorMultiLevel report from node 9, instance 1, Temperature:
value=78F
Received SensorMultiLevel report from node 9, instance 1, Luminance:
value=33%
Node 9 has been marked as awake
Node 9 has been marked as asleep
Received SensorBinary report: Sensor:1 State=Off
```

Konfigurační soubor obsahuje množinu pravidel pro vyhodnocování těchto událostí, kde se každé pravidlo skládá z RV události, názvu MIB objektu, kam se bude ukládat, a RV pro získání hodnoty objektů. Tyto tři součásti jsou oddělené středníkem ve formátu `csv`. Příklad navrženého pravidla vypadá následovně:

```
Received SensorMultiLevel report from node \d{1,2},instance \d{1,2},
Luminance: value=\d{1,3}\%; sensorLuminance;value=(.+?)\%
```

Pravidlo je určeno pro získaní hodnoty svítivosti u dveřního senzoru. Řádek obsahuje tři části oddělené středníkem, kde první část je RV, který odpovídá události:

```
2020-02-05 21:59:03.599 Info, Node009, Received SensorMultiLevel report
from node 9, instance 1, Luminance: value=33%
```

Druhá část je název MIB objektu *sensorLuminance*, do kterého se uloží hodnota získaná pomocí regulárního výrazu:

```
value=(.+?)\%
```

Vytvořený konfigurační soubor je gramatikou pro analyzátor. Podstatou algoritmu analyzátoru je na základě definovaných pravidel procházet události. Pokud se našla událost, která je novější, než čas uložený ve sdílené paměti, vyhodnotit hodnotu objektu z nalezeného řádku a tuto hodnotu uložit do databáze MIB. Parser je implementovaný v jazyku Python.

Pro optimalizaci parseru a zajištění aktuálních hodnot byl vytvořen skript v bashi, který vybírá z logovacího souboru jen nejnovější události, které jsou pak vstupem pro parser.

Skript prochází logovací soubor, pokud najde řádek, který má starší časovou značku, přepíše časovou značku na nejnovější čas a starší události vyhodí. Připraví nový soubor

[5]Seznam tříd příkazů Z-Wave, viz http://wiki.micasaverde.com/index.php/ZWave_Command_Classes

události, který je vstupem pro analyzátor. Kód lze najít v příloze E. Pseudokód parseru bude pak vypadat následovně:

```
schema = [
    {'mib_object': 'sensorLuminance',
        'mib_oid': '1.3.6.1.4.1.8072.2.4.1.1.4.0',
        'mib_value': ''},
    {'mib_object': 'sensorTemperature',
        'mib_oid': '1.3.6.1.4.1.8072.2.4.1.1.5.0',
        'mib_value': ''},
    {'mib_object': 'sensorState',
        'mib_oid': '1.3.6.1.4.1.8072.2.4.1.1.3.0',
        'mib_value': ''},
    {'mib_object': 'sirenState',
        'mib_oid': '1.3.6.1.4.1.8072.2.4.1.1.2.0',
        'mib_value': ''},
]

udalosti = open(file, 'r')

for radek in udalosti:
    aktualni_cas = vratit_aktualni_cas()
    for pravidlo in pravidla:
        udalost, prikaz = najdi_udalost(radek), najdi_prikaz(radek)
        if udalost and prikaz:
            for prvek in schema:
                if prvek['mib_object'] == pravidlo['mib_object']:
                    if aktualni_cas >= posledni_cas_udalosti:
                        prvek['mib_value'] = zjisti_hodnotu_objektu(radek)
                        posledni_cas_udalosti = aktualni_cas
```

Nové události parser ukládá do MIB databáze za použití knihovny `pysnmp.hlapi`[6]. Implementace funkci *get*, *set* byla převzata z tutoriálu Python SNMP[7]. Níže je uveden kód pro uložení nových hodnot do MIB databáze.

```
hostname = '192.168.251.2'
community = CommunityData('public')
for item in schema:
    set(hostname, {item['mib_oid']: item['mib_value']}, community)
```

5.5 Shrnutí

V této kapitole jsme se podívali, jak vytvořit vlastní MIB databázi, kterou jsme vložili do experimentálního stromu. Popsali jsme objekty *sirenState* a *sensorState*, které nám pomohou při monitorování zařízení.

[6] Viz http://snmplabs.com/pysnmp/docs/pysnmp-hlapi-tutorial.html
[7] Viz https://www.ictshore.com/sdn/python-snmp-tutorial/

Implementovali jsme rozšiřitelného agenta za použití protokolu AgentX, podívali jsme se, jak agent přijímá požadavky přicházející od manažera. Také jsme vygenerovali moduly pro registraci objektů a jejich zpřístupnění pro manažera.

Dále byl zmíněn formát zpráv z IoT komunikace a způsob jejich interpretace pomocí dvou skriptů a následné ukládání nových hodnot do MIB databáze.

V příloze C je ukázáno, jak spustit hlavního agenta a zkompilovat subagenta s vytvořenými objekty pro monitorování. Příloha D obsahuje vytvořenou MIB databázi a bash skript pro zpracování logovacích informací. Definice regulárních výrazu lze najít v příloze E.

V další kapitole bude popsáno testování funkčnosti agenta, který bude přijímat požadavky na získání hodnot nově vytvořených objektů.

Kapitola 6

Testování

V práci byl vytvořen systém SNMP, který poskytuje monitorovací funkci zařízením komunikujícím přes rádiový signál Z-Wave. Ve virtuálním prostředí běžel agent a rozšířený agent. Zároveň na počítači mimo virtuální prostředí běžel nástroj Home Assistant. K počítači byl také připojený USB-vysílač přes V/V sběrnici, který naslouchal na Z-Wave síti. Jednou za minutu se spouštěl parsovací skript. Při experimentech se SNMP manažer dotazoval agenta na hodnoty objektů zařízení. Bylo vidět, jak se mění hodnoty objektů zařízení, například teplota a svítivost, stav (asleep/awake) u dveřního senzoru a stav (On/Off) u sirény. Následující tabulka obsahuje zvolené MIB objekty pro monitorování:

Jméno objektu	OID	Syntax
sirenState	1.3.6.1.4.1.8072.2.4.1.1.2.0	DisplayString (SIZE(0..3)
sensorState	1.3.6.1.4.1.8072.2.4.1.1.3.0	DisplayString (SIZE(0..6))
sensorLuminance	1.3.6.1.4.1.8072.2.4.1.1.4.0	Integer32
sensorTemperature	1.3.6.1.4.1.8072.2.4.1.1.5.0	Integer32

Test 1

Logovací soubor obsahuje událost „stav dveřního senzoru“:

```
2020-02-05 21:59:05.122 Info, Node009, Node 9 has been marked as awake
```

kterou SNMP agent zpracoval jako:

```
$ snmpget -v2c -c public 192.168.251.2 1.3.6.1.4.1.8072.2.4.1.1.3.0
NET-SNMP-EXAMPLES-MIB::netSnmpExamples.4.1.1.3.0 = STRING: "awake"
```

Test 2

Později došlo ke změně hodnoty události:

```
2020-02-05 21:59:05.174 Info, Node009, Node 9 has been marked as asleep
```

která se hned uložila do databáze:

```
NET-SNMP-EXAMPLES-MIB::netSnmpExamples.4.1.1.3.0 = STRING:
"asleep"
```

Test 3

Logovací soubor obsahuje událost „teplota senzoru“:

```
2020-02-05 21:59:03.573 Info, Node009, Received SensorMultiLevel report
from node 9, instance 1, Temperature: value=78F
```

agent pak uložil hodnotu následovně:

```
NET-SNMP-EXAMPLES-MIB::netSnmpExamples.4.1.1.5.0 = INTEGER: 78
```

Test 4

Logovací soubor obsahuje událost „svítivost senzoru“, která vypadá následovně:

```
2020-02-05 21:59:03.599 Info, Node009, Received SensorMultiLevel report
from node 9, instance 1, Luminance: value=33%
```

do databáze se pak uložila hodnota:

```
NET-SNMP-EXAMPLES-MIB::netSnmpExamples.4.1.1.4.0 = INTEGER: 33
```

Test 5

Ukázka události „stav sirény“

```
2020-02-11 14:34:23.336 Info, Node012, Received SensorBinary report:
Sensor:1 State=Off
```

kterou SNMP agent zpracoval jako:

```
NET-SNMP-EXAMPLES-MIB::netSnmpExamples.4.1.1.2.0 = STRING: "Off"
```

Test 6

Dále došlo ke změně hodnoty události:

```
2020-02-11 14:34:23.345 Info, Node012, Received SensorBinary report:
Sensor:1 State=On
```

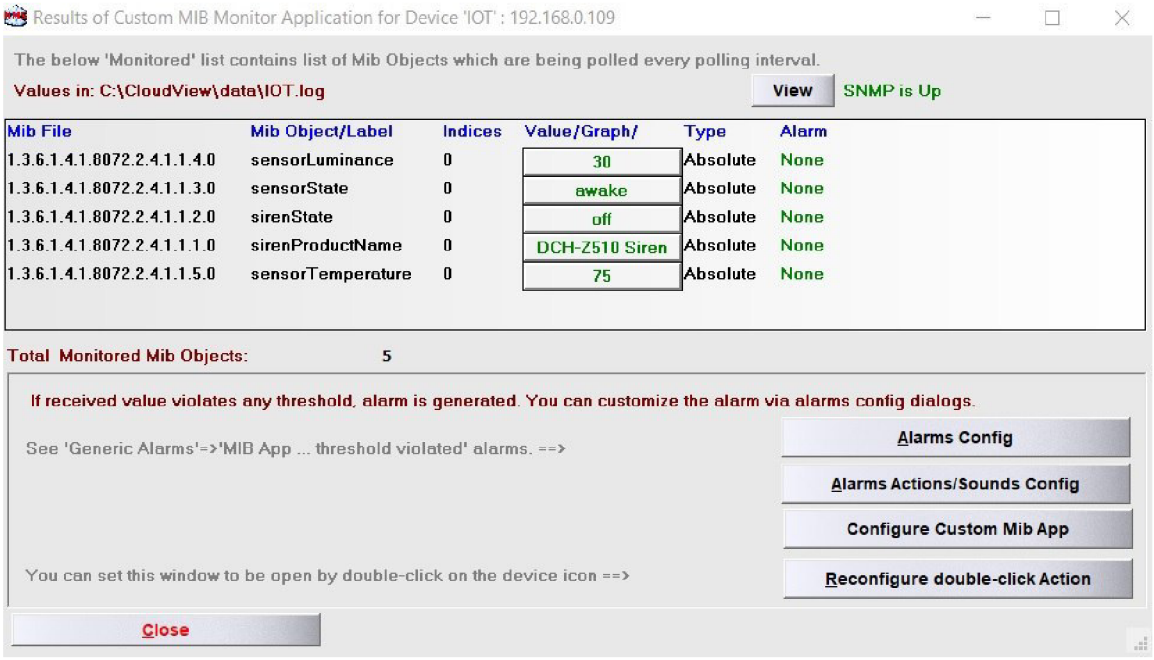
která se následně zpracovala jako:

```
NET-SNMP-EXAMPLES-MIB::netSnmpExamples.4.1.1.2.0 = STRING: "On"
```

6.1 SNMP manager

V současné době existuje sada nástrojů pro monitorování a sledování stavu různých služeb počítačové sítě, serverů a síťových zařízení. Mezi nástroji, které mají grafické rozhraní, jsou MRTG[1], PRTG[2], Zabbix[3], Nagios[4], CloudView NMS[5], LibreNMS[6] a další.

V práci byl zvolen SNMP manager CloudView NMS. NMS server pravidelně dotazoval SNMP agenta na vybrané OID s IoT objekty a zobrazoval jejich hodnotu. Konfigurace manažera je popsaná v příloze F. Ukázka monitorování hodnot objektů je na obrázku 6.1.



Obrázek 6.1: Ukázka monitorování hodnot MIB objektů

6.2 Shrnutí

V této části bylo popsáno monitorování SNMP objektů na jednoduché IoT sítí. Systém zpracovával události z IoT komunikace, a když docházelo ke změně hodnoty události, program ukládal nové hodnoty do MIB databáze. Během provozu SNMP periodicky načítalo nové hodnoty zvolených SNMP objektů. Dále byl ukázán jednoduchý nástroj CloudView NMS s podporou grafického rozhraní pro monitorování.

[1] Viz https://oss.oetiker.ch/mrtg/
[2] Viz https://www.paessler.com/prtg
[3] Viz https://www.zabbix.com/
[4] Viz https://www.nagios.org/
[5] Viz http://www.cloudviewnms.com/
[6] Viz https://www.librenms.org/

Kapitola 7

Závěr

IoT zařízení jsou jednoduchá zařízení, která jsou připojena k síti (včetně internetu), nebo k jiným strojům a pracují samostatně bez nutnosti lidského zásahu. Většinou IoT zařízení jsou připojeny pouze přes L2 technologie (Bluetooth, Zigbee, Z-Wave), což neumožňuje monitorovat IoT síť standardním způsobem, a je proto potřeba vytvořit speciální přístup.

Systém SNMP umožňuje správu sítě a průběžný sběr nejrůznějších dat o zařízeních. Je to standard sledovaní řízení bezpečnosti sítě, zpracování chyb a odhalení možných útoků. Některá IoT zařízení tento systém však nepodporují, neboť nepoužívají TCP/IP, ale komunikují přes L2 technologie. Přínosem této práce je rozšíření Z-Wave komunikace o systém SNMP. Tento nástroj je důležitý pro organizace používající IoT zařízení pro zkoumání stavu zařízení na útoky, aby je bylo možné včas odhalit.

Tato bakalářská práce popisuje postup vytvoření SNMP agenta v jazyku C, MIB databáze a navrhuje analyzátor, který interpretuje data z logovacího souboru na hodnoty proměnných v databázi MIB. Daný postup lze aplikovat i na jiné IoT sítě, které lokálně používají rádiové spojení, například typu Bluetooth, Zigbee apod. Je potřeba pouze adaptér pro danou síť a softwarový nástroj Home Assistant pro zpracování události. Dále je potřeba popsat události, které se budou odchytávat pomocí regulárních výrazů a vytvořit definici objektů MIB.

Daná práce byla zaslaná a přijata na studenskou soutěž Excel@FIT.

Literatura

[1] AgenPro. Naposledy navštíveno 26. 5. 2020. Dostupné z: <https://agentpp.com/tools/agenpro.html>

[2] Agentuino. Naposledy navštíveno 26. 5. 2020. Dostupné z: <https://code.google.com/archive/p/agentuino/>

[3] Case, J.; aj. : A Simple Network Management Protocol (SNMP). IETF RFC 1157, 1990.

[4] Davin, J.; aj. : A Simple Gateway Monitoring Protocol. IETF RFC 1157, 1987.

[5] Dundar, O. : *Home Automation with Intel Galileo.* 3. vydání. Packt Publishing Ltd, 2015.

[6] ITU-T : X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). Recommendation I.209, 1988.

[7] Matoušek, P. : *Síťové aplikace a jejich architektura.* Brno, Česká republika: VUTIUM, 2014. 341-350 s.

[8] McCloghrie, K.; aj. : Structure of Management Information Version 2 (SMIv2). IETF RFC 1157, 1999.

[9] Rose, M.; McCloghrie, K. : Structure and Identification of Management Information for TCP/IP-based Internets. IETF RFC 1155, 1990.

[10] Stallings, W. : *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* 3. vydání. Addison-Wesley, 1998.

[11] Zeman, O. *Implementace simulačního modelu zjednodušené databáze DiffServ-MIB [online]* Brno: VUT, 2008.

Příloha A

Instalační prostředí

V této příloze je popsaná konfigurace systému SNMP za použití operačního systému Ubuntu 18.04 LTS[1], nástroje Net-SNMP 5.8 a také nástroje Home Assistant pro monitorování Security Kit objektů. Nástroj je k dispozici ke stažení na oficiálních stránkách.[2].

A.1 Instalace systému SNMP

Před instalací systému SNMP je nutné nainstalovat požadované balíčky příkazem:

```
$ sudo -i && apt-get install build-essential && apt-get install libperl-dev
```

Nejjednodušší způsob instalace net-snmp je:

1. Následující příkazy budou spouštěny pomocí privilegovaného režimu `sudo`:

   ```
   $ sudo -i
   ```

2. `cd` do adresáře obsahujícího zdrojový kód a `./configure` pro konfiguraci balíčku net-snmp. Po spuštění příkazů se detekují potřebné balíčky (gcc, egrep, sys/types.h atd) a provádějí se určité kontroly, např. zda funguje compilator `gcc`, zda `gcc` podporuje ISO C89 atd. Skript shellu `configure` se pokouší uhodnout správné hodnoty různých systémových proměnných, které byly použity po dobu kompilace. Tyto hodnoty se použijí k vytváření *Makefile* souboru v každé složce balíčku. Skript vytváří několik '*.h' souborů obsahujících definice závislé na systému. Během konfigurace je uživatel vyzván k sérii otázek, které rozhodnou o nastavení snmp. V tuto chvíli se generuje také soubor `include/net-snmp/net-snmp-config.h`. Obsahuje důležité informace, jako je umístění logů a konfiguračních souborů.

3. Pro kompilaci balíčku zadejte `make`

4. Zadejte `make test`, který spustí řadu testů, abyste viděli, jaká funkcionalita byla zahrnutá a zda funguje.

5. Zadejte `make install` pro instalaci programů a všech datových souborů a dokumentace.

[1] Viz https://ubuntu.com/download/desktop
[2] Viz http://www.net-snmp.org/download.html

6. Můžete odstranit binární soubory programu a soubory objektů z adresáře zdrojového kódu zadáním `make clean`. Chcete-li také odstranit soubory, které `configure` vytvořil, napište `make distclean`.

7. Aplikaci můžete odebrat zadáním `make uninstall`.

Konfigurační soubory se nacházejí ve složce *$prefix*`/share/snmp`, kde *$prefix* je definován jako předaná hodnota do argumentu `--prefix` konfiguračního skriptu, nebo `/usr/local`, pokud nebylo definováno. Pro konfiguraci byla mnou zvolená defaultní konfigurační složka `/usr/local/share/snmp`.

A.2 Konfigurace agenta

Provadí se příkazem `snmpconf`.

1. Zvolte typ souboru, který chcete vytvořit: *snmpd.conf*
2. Zvolte Access Control Setup.
3. Zvolte nastavení SNMPv1/SNMPv2c s povolením jen pro čtení:
   - community name: public
   - network address to accept: localhost
   - restricted OID [RETURN for no-restriction]:
4. Zvolte nastavení SNMPv1/SNMPv2c s povolením pro čtení a zápis:
   - community name: private
   - network address to accept: 127.0.0.1
   - restricted OID [RETURN for no-restriction]:
5. finished, finished, quit
6. Nakopírujte vytvořený konfigurační soubor do konfigurační složky:

   ```
   $ sudo cp snmpd.conf /usr/local/share/snmp/
   ```

7. Pro vzdáleny přístup je potřeba do konfiguračního souboru přidat také řádky:

   ```
   agentAddress udp::161
   rwcommunity public 192.168.251.1
   view all included 1.
   ```

   kde 192.168.251.1 je IP adresa hosta, kterému chceme povolit přistup.

8. `$ export LD_LIBRARY_PATH=/usr/local/lib`

Příklad konfiguračního souboru by pak mohl vypadat následovně:

```
agentAddress udp::161
rwcommunity public localhost
rwcommunity private 127.0.0.1
rwcommunity public 192.168.251.1
master agentx
view all included .1
```

A.2.1 Spuštění a testování agenta

Ve složce net-snmp-5.8/agent je potřeba vytvořeného agenta spustit za privilegovaného režimu:

```
$ sudo ./snmpd
```

Příkaz *snmpget* lze použít k načtení dat ze vzdáleného hosta podle názvu hosta, autentizačních informací a OID, viz následující příklad:

```
$ snmpget -v2c -c public localhost system.sysUpTime.0
system.sysUpTime.0 = Timeticks: (586731977) 67 days, 21:48:39.77
```

Výše uvedený příklad vrací hodnotu OID *system.sysUpTime.0*, kde *localhost* je adresa SNMP agenta a *public* je název community.

A.2.2 Konfigurace subagenta

Subagenta lze spustit a připojit k hlavnímu agentovi *snmpd*. Před spuštěním master agenta *snmpd* je nutné do souboru *snmpd.conf* v konfigurační složce vložit řádek `master agentx` pro zapnutí podpory AgentX. Pokud jste tak ještě neučinili, musíte zastavit agenta, přidat konfigurační řádek a restartovat agenta snmpd.

Příloha B

Instalace Home Assistant

Pro instalaci virtuálního prostředí je potřeba mít nainstalovaný Python 3.7. Po instalaci Pythonu je potřeba nainstalovat Home Assistant a přidat podporu Z-Wave.

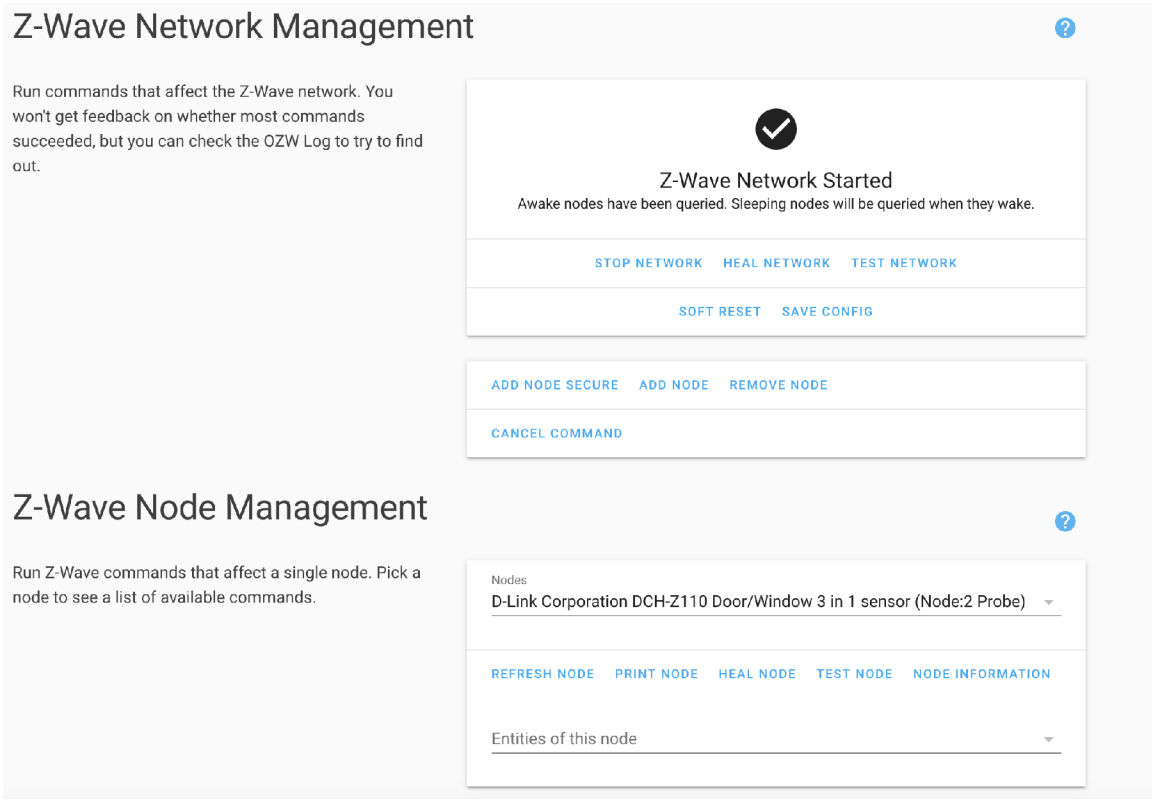
```
$ pip3 install homeassistant
$ cd homeassistant/homeassistant && source bin/activate
$ pip install python_openzwave
```

Přes sběrnici je potřeba připojit USB-vysílač a nakonfigurovat správný název zařízení v souboru `/.homeassistant/configuration.yaml`. Zařízení implementuje virtuální sériové rozhraní používané aplikacemi Z-Wave. Linux a Mac OSX mají vestavěný ovladač zařízení a vytváří nové zařízení s názvem jako */dev/cu.usbmodem142101* (OSX) nebo */dev/ttyACM0* (Linux). Windows může vyžadovat ovladač zařízení *uzb.inf* na adrese www.z-wave.me.

Konfigurační soubor pak může vypadat následovně:

```
default_config:

tts:
  - platform: google_translate

group: !include groups.yaml
automation: !include automations.yaml
script: !include scripts.yaml
zwave:
    usb_path: /dev/cu.usbmodem142101
```

Příkaz `hass --open-ui` spustí Home Assistant ve webovém rozhraní *http://localhost:8123*. V kontrolním panelu pak lze najít za cestou Configuration → Z-Wave panel správy uzlu Z-Wave.



Obrázek B.1: Rozhraní Home Assistant

Příloha C

Uživatelské rozhraní

Když jsou agent a nutné moduly implementovány, je potřeba agenta spustit a ukázat na jednoduchém příkladu funkčnost aplikace.

C.1 Kompilace

Vytvořeného subagenta SNMP a moduly pro registraci objektů je nutné zkompilovat do snmp-daemona pomocí příkazu `make snmp-daemon`.

```
CC=gcc

OBJS1=snmp-daemon.o sirenProductName.o sirenState.o
sensorState.o sensorLuminance.o sensorTemperature.o

CFLAGS=-I. `net-snmp-config --cflags`
BUILDAGENTLIBS=`net-snmp-config --agent-libs`

# shared library flags (assumes gcc)
DLFLAGS=-fPIC -shared

snmp-daemon: $(OBJS1)
       $(CC) -o snmp-daemon $(OBJS1) $(BUILDAGENTLIBS)
```

C.2 Spuštění agenta a manažera SNMP

Agent a subagent se spouští pomocí příkazu:

```
$ cd net-snmp5.8/agent && sudo ./snmpd
$ sudo ./snmp-daemon
```

Zkusíme otestovat subagenta pomocí SNMP manažera:

```
$ snmpget -v2c -c private 127.0.0.1 IOT-MIB::sirenState.0
IOT-MIB::sirenState.0 = STRING: off
```

Příloha D

MIB databáze

```
IOT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    netSnmpExamples                    FROM NET-SNMP-EXAMPLES-MIB
    OBJECT-TYPE, Integer32,
    MODULE-IDENTITY                        FROM SNMPv2-SMI
    DisplayString                          FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP        FROM SNMPv2-CONF;

iotMIB MODULE-IDENTITY
    LAST-UPDATED "200205290000Z"            -- 22 November 2019, midnight
    ORGANIZATION "VUT"
    CONTACT-INFO "postal:   Kateryna Polishchuk
                  email:    xpolis03@fit.vutbr.cz"
    DESCRIPTION  "A mib for monitoring D-Link Corporation DCH-Z510 Siren."
    ::= { netSnmpExamples 4 }

iotMIBObjects     OBJECT IDENTIFIER ::= { iotMIB 1 }
iotAgentModules   OBJECT IDENTIFIER ::= { iotMIBObjects 1 }

sirenProductName OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
                "This is an object that provides information
                about product name of device"
    DEFVAL { "DCH-Z510 Siren" }
    ::= { iotAgentModules 1 }

sirenState OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..3))
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
```

```
        "Status of the device may take on values on/off"
    DEFVAL { "off" }
    ::= { iotAgentModules 2 }

sensorState OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..6))
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Door/Window Sensor - asleep/awake"
    DEFVAL { "asleep" }
    ::= { iotAgentModules 3 }

sensorLuminance OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Door/Window luminance"
    DEFVAL { 0 }
    ::= { iotAgentModules 4 }

sensorTemperature OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Door/Window temperature"
    DEFVAL { 0 }
    ::= { iotAgentModules 5 }
END
```

Příloha E

Zpracování události

```
#!/bin/bash

udalosti="OZW_Log.txt"
out="out.txt"
last_time_file="times.txt"
datetime_regex='([0-9][0-9][0-9][0-9])-([0-9][0-9])-([0-9][0-9])
 ([0-9][0-9]:[0-9][0-9]:[0-9][0-9].[0-9][0-9][0-9])'

last_time=$(head -n 1 "times.txt")
last_time=$(gdate --date="$last_time" +'%Y-%m-%d %H:%M:%S.%6N');

FROM_LINE_NUMBER=0
while IFS= read -r line
do
    let "FROM_LINE_NUMBER++"
    if [[ $line =~ $datetime_regex ]]; then
        current_time="${BASH_REMATCH[1]}-${BASH_REMATCH[2]}-
        ${BASH_REMATCH[3] ${BASH_REMATCH[4]}"
        time=$(gdate --date="$current_time" +'%Y-%m-%d %H:%M:%S.%6N');
        if [[ "$time" > "$last_time" ]] ;
        then
           > "$last_time_file"
           echo "$time" >> "$last_time_file"
           echo "$FROM_LINE_NUMBER"
           echo "break"
           break
        fi
    fi
done < "$udalosti"
echo "$FROM_LINE_NUMBER"
> "$out"
tail -n +"$FROM_LINE_NUMBER" "$udalosti" >> "$out"
```

Příloha F

Konfigurace CloudViewNMS

CloudViewNMS[1] je nástroj pro monitorování síťových prvků, který má jednoduché rozhraní a umožňuje monitorovat prvky podle OID. Po stáhnutí a instalaci aplikace postup konfigurace je následující:

- Přidat nové zařízení v horním menu aplikace s typem *Generic Devices* – `GENERIC` - Generic SNMP Device. Dále nastavit IP adresu zařízení, na kterém běží agent, verzi SNMP protokolu, název comunity (například public/private), interval dotazovaní.
- Přidat OID sledovaných objektů kliknutím pravého tlačítka na zařízení: SNMP → Build Custom MIB Monitoring App → Add Raw OID. Dále zadat OID objektu, který chceme monitorovat, zjistit hodnotu pomocí tlačítka `SNMP-GET the above object` a uložit. Zadat všechny OID objektů pro monitorování.
- Stisknutím pravého tlačítka na zařízení zvolit SNMP → Results of Custom MIB Monitoring App. V otevřeném okenku monitorujeme hodnoty objektů.

[1] Viz http://www.cloudviewnms.com/