



Fakulta elektrotechnická
Katedra elektroenergetiky a ekologie

BAKALÁŘSKÁ PRÁCE

Návrh a realizace real-time komunikace pro senzorickou síť
s webovou řídicí aplikací

Design and Implementation of Real-time Communication for
Sensory Network with Website Based Control Application

Autor práce: Martin Zlámal
Vedoucí práce: Ing. Petr KRIST, Ph.D.

Plzeň 2014

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Martin ZLÁMAL**
Osobní číslo: **E13B0267P**
Studijní program: **B2612 Elektrotechnika a informatika**
Studijní obor: **Technická ekologie**
Název tématu: **Návrh a realizace real-time komunikace pro senzorickou síť s webovou řídicí aplikací**
Zadávající katedra: **Katedra elektroenergetiky a ekologie**

Z á s a d y p r o v y p r a c o v á n í :

1. Prostudujte si a teoreticky zpracujte dostupné materiály k problematice real-time komunikací pro inteligentní senzorické sítě s přihlédnutím k webovým aplikacím. Seznamte se s hardwarovými prostředky senzorické sítě.
2. Na základě předchozího bodu zvolte vhodnou technologii a navrhnete strukturu programového řešení v závislosti na dostupném hardware.
3. Napište program obsluhující komunikační jednotky senzorické sítě a naprogramujte webovou aplikaci pro ovládání této sítě.
4. Rozeberte možnosti praktické aplikace této sítě a její možnosti rozšíření.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **20 - 30 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí bakalářské práce: **Ing. Petr Krist, Ph.D.**


Katedra aplikované elektroniky a telekomunikací

Datum zadání bakalářské práce: **15. října 2014**

Termín odevzdání bakalářské práce: **8. června 2015**


Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan




Doc. Ing. Karel Noháč, Ph.D.
vedoucí katedry

V Plzni dne 15. října 2014

Abstrakt

Text abstraktu v češtině...

Klíčová slova

Ethernet, Expres.js, Node.js, Procesor, Redis, RESP, TCP, UDP, Websocket

Abstract

Text abstraktu v angličtině...

Key Words

Ethernet, Expres.js, Node.js, Procesor, Redis, RESP, TCP, UDP, Websocket

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Obsah

Seznam obrázků	vi
Seznam symbolů a zkratek	vii
1 Úvod	1
2 Real-time komunikace	2
2.1 Hardwarové prostředky senzorké sítě	3
2.2 Real-time ve webových aplikacích	4
2.3 TCP	5
2.4 UDP	6
3 Volba vhodné technologie	8
3.1 Prvky senzorké sítě	8
3.1.1 Procesor STM32F207IGH6	8
3.1.2 Procesor STM32F457IGH6	9
3.2 Real-time server	9
3.3 Databázový server	10
3.3.1 Redis klíče	10
3.3.2 Datová struktura string	11
3.3.3 Datová struktura hash	11
3.3.4 Datová struktura list	11
3.3.5 Datová struktura set a sorted set	12
3.3.6 Datová struktura bitmap	12
3.3.7 Datová struktura hyperloglog	12
3.3.8 RESP protokol	13
3.4 Webová aplikace	14
4 Struktura programového řešení	15
4.1 Komunikace koncentrátor - server	15
4.2 Komunikace server - webová aplikace	16

5	Praktická aplikace	17
6	Rozšíření stávajícího řešení	18
7	Závěr	19

Seznam obrázků

2.1	Uspořádání TCP packetu	6
2.2	Uspořádání UDP datagramu	7

Seznam symbolů a zkratek

AJAX	Asynchronous JavaScript and XML
AJAJ	Asynchronous JavaScript and JSON
BGA	Ball Grid Array
EDA	Event-Driven Architecture
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
JSONP	JSON with padding
MAC	Media Access Control
RAM	Random-Access Memory
RESP	Redis Serialization Protocol
RFC	Request for Comments
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UTP	Unshielded Twisted Pair
UFBGA	Ultra Fine BGA
XHR	XMLHttpRequest
XML	Extensible Markup Language

1

Úvod

Cílem této práce je navrhnout komunikaci pro senzorickou síť s přihlédnutím k tomu, že by tato síť měla být ovladatelná real-time z webové aplikace. Toto je velmi zásadní požadavek pro budoucí realizaci, protože z hlediska elektronických systémů je real-time komunikaci možné realizovat pomocí protokolů k tomu určených, které vymezují přenos dat do přesně definovaných časových slotů (Ethernet Powerlink, Time-triggered CAN, FlexRay). U webových aplikací žádný takový prvek neexistuje a webová řídicí aplikace se tak stává limitujícím prvkem celé sítě. Existují však metody, které se real-time komunikaci resp. rychlé komunikaci, jak je real-time u webových aplikací všeobecně chápán, mohou velmi přiblížit. V roce 2011 bylo vydáno RFC 6455 [1], které zastřešuje nový protokol websocket, který umožňuje propojení serveru a klientské části aplikace socketem a je tak možné přenášet informace velmi vysokou rychlostí, což doposud nebylo prakticky téměř možné realizovat.

V následující části práce bude rozebrána problematika komunikace senzorické sítě s webovou řídicí aplikací, ze které vyplyne, že nejvhodnějším řešením je naprogramovat jednotlivé členy senzorické sítě co nejvíce nízkoúrovňově, následně je propojit s řídicím serverem, na kterém poběží Node.js real-time server (asynchronní single-thread) pro zpracovávání požadavků a zároveň zde poběží server pro webovou aplikaci, která bude využívat websocket protokolu coby nástroje pro komunikaci s tímto serverem. Zároveň je tato senzorická síť uváděna na příkladu administrativní budovy resp. jakéhokoliv objektu kde se běžně pohybují lidé a využívají konvenční elektroinstalaci, tzn. například domácí objekty, popřípadě jiné objekty podobného charakteru kde má využití této sítě praktický přínos.

2

Real-time komunikace

Real-time komunikace představuje významný prvek v aplikacích, kde je zapotřebí přesných časových rámování přenášeného signálu. Real-time systém je buď hardwarový nebo softwarový systém, který by měl komunikovat s řídicím systémem v přesně stanovených časových periodách. Tato definice tedy neznamená, že by měl systém odpovídat, nebo posílat data okamžitě, ale že garantuje reakci systému v daném časovém intervalu a to buď reakcí na výzvu od řídicího signálu, nebo ve fixní časy (tedy v relativní nebo absolutní čas) [2]. Dále lze real-time komunikaci rozdělit na tzv. soft real-time a hard real-time. Rozdíl je pouze v samotném přístupu ke spolehlivosti přenosu informace. U soft real-time přístupu je možné připustit, že se informace po nějakém čase zahodí, jelikož je již nežádoucí. U hard real-time toto není možné.

Obecně lze však za real-time aplikaci uvažovat systém, který reaguje na požadavky bez zbytečného dopravního zpoždění, které je například u webových aplikací naprosto běžné a odezvy v přesně definovaných časových odezvách se nepoužívají zejména z důvodu rychlosti. Je totiž mnohem důležitější odeslat data se serveru co nejrychleji, než je posílat podle časově definovaných oken. Předejít však dopravnímu zpoždění u webových aplikací není možné. Důvod je prostý. Webová aplikace musí být dostupná pro všechny uživatele na celém světě a z toho plyne, že každý uživatel je na jiném geografickém místě a čas potřebný k dostání informace ke koncovým uživatelům není stejný. Tento problém lze částečně vyřešit distribuovaným systémem, kdy se servery přibližují uživatelům, což prakticky dělají například streamovací portály jako je YouTube. Toto řešení má svá omezení a proto druhým způsobem, jak ušetřit čas při komunikaci s koncovým prvkem, je zjednodušit komunikační protokol, nebo se omezit na co nejméně zbytečné režie a to i za tu cenu, že nedojde ke stoprocentnímu přenosu informace.

2.1 Hardwarové prostředky senzorké sítě

Hardwarové prostředky této sítě nejsou v současné chvíli nijak přesně definovány. Je tedy možné síť navrhnout libovolným způsobem. Vzhledem ke komplikovanosti celé problematiky bude tato síť striktně metalická paketová. Taková síť se tedy skládá v nejmenší konfiguraci pouze z koncového členu a serveru. S narůstajícím počtem koncových členů je zapotřebí síť patřičně rozšiřovat. Výhodou tohoto systému je fakt, že se daná síť nijak neliší od běžných metalických ethernetových sítí, tzn. že lze využít veškeré dostupné prostředky pro tvorbu této sítě a není zapotřebí vyvíjet zbytečně drahá nová zařízení.

Celá síť se tak skládá z klasického ethernetového vedení a rozbočovačů, přepínačů popř. směrovačů. Zbývá tedy vyřešit server a koncové členy. Zde však záleží na praktické aplikaci. Vezmeme-li však v úvahu nejobyčejnější systém, server pak může být prakticky jakýkoliv počítač, který dokáže zpracovat příchozí požadavky. Tzn. musí být dostatečně výkonný a pro lepší bezpečnost celého systému také redundantní (nebo alespoň některé kritické komponenty v něm). Redundanci komponent však dobře řeší klasické servery, kde jsou redundantní například zdroj, pevné disky, řadiče a dále duální paměti popř. procesory.

Samotné koncové prvky se pak sestávají z nízkoodběrových procesorů, které mají menší, pro danou aplikaci však dostatečný výkon. Zde opět záleží na daném účelu koncového zařízení. Pokud má sloužit jako koncentrátor, tedy zařízení sbírající data ze senzorů, potřebuje větší výkon než například termální čidlo. Výkon koncového prvku je tak dán samotným programem, který na tomto prvku poběží.

Tato síť je tedy v takovém stavu, kdy je zapojen server (nejlépe na nezávislém napájení) a senzory jsou zapojeny v ethernetové síti pomocí běžných síťových prvků. Důležité je však vyřešit co se stane, když vypadne napájení? V tomto okamžiku síť prakticky přestane fungovat. Toto se nijak neliší od např. běžné zapojení elektroinstalace. Sice by šlo zajistit napájení koncových prvků, protože server může být zapojen na více nezávislých zdrojích elektrické energie, to však nebude např. v rodinném domě běžné. Horší případ nastane, když vypadne připojení k internetu. Zde by se nejednalo o problém, pokud by se server nacházel v řízeném objektu. Jediný efekt by byl ten, že by nebylo možné server ovládat vzdáleně. Horší situace ovšem nastane v okamžiku, kdy je server umístěn ve vzdálené serverovně. V takovém případě je pro tuto senzorkou síť potřeba vyřešit tzv. disaster solution, tedy nějaké fallback zařízení při selhání. Samotné koncové členy musí vědět jak se chovat bez příchozího signálu. To většinou není problém, protože paradoxně není potřeba řešit jejich chování. To je nutné pouze v případě zabezpečení ob-

jektů. Starostí koncových členů totiž není např. vypnout světlo, pokud není systém připojen k internetu. V takovém objektu je však zapotřebí zařadit do sítě zařízení, které bude přijímat od serveru povely a obsluhovat síť. V případě přerušení spojení se serverem převezme toto zařízení kontrolu nad sítí a uvede objekt do dočasného módu, než se problém vyřeší, nebo než přijede servis. Bude tak možné i nadále ovládat alespoň na základní úrovni většinu zařízení.

2.2 Real-time ve webových aplikacích

Ve webových aplikacích žádný real-time jako takový v podstatě neexistuje. Existují však technologie, které umožňují rychlou komunikaci s webovým serverem, resp. rychlou výměnu dat, což vždy nemusí být jedno a to samé.

Jedním z typických zástupců je AJAX (popř. AJS). Jedná se jednosměrný mechanismus, kdy se po periodické akci, nebo například při stisku tlačítka vyvolá javascriptová akce, která uzavře HTTP spojení se serverem a získá data v závislosti na požadavku. Následně překreslí část stránky obsahující nová data. Nedojde tak k obnovení celé stránky, ke kterému by došlo při běžném pohybu návštěvníka na stránce. Výhodou je, že není zapotřebí přenášet celou stránku. Nevýhodou však je možný nárůst HTTP požadavků na server a hlavně nutnost vyjednat se serverem spojení při každém požadavku, což je časově velmi náročné. Pro tuto aplikaci je proto použití AJAXu nevhodné.

Oproti tomu websocket [1] je protokol, který umožňuje otevřít socket mezi serverem a prohlížečem a pomocí rámců posílat obousměrně informace. Vyjednat spojení se serverem tak stačí pouze jednou při otevření webové stránky a následně je možné velmi rychle se stránkou komunikovat. Zároveň se periodicky kontroluje, jestli je stránka stále aktivní (tzv. heartbeat) a pokud ne, server spojení uzavře. Nespornou výhodou je také fakt, že websocket využívá principu event-driven, takže kromě periodické kontroly aktivního spojení je možné posílat data pouze pokud je to nutné, což hodně ušetří na komunikaci mezi serverem a browserem. Websocket staví nad HTTP, takže mu dnešní prohlížeče rozumí, nicméně pro případ toho, že by webovou stránku otevřel uživatel ve starším prohlížeči, jsou většinou k dispozici fallback řešení ve formě jiných technologií tak, aby stránka fungovala. V tomto případě se jedná zejména o XHR-polling a JSONP-polling.

2.3 TCP

Protokol TCP je jedním ze dvou transportních protokolů [3], které tento systém využívá. Stejně tak jako UDP je zde tento protokol rozbírán zejména z toho důvodu, že právě na TCP paketech a UDP datagramech je vystavěna komunikace mezi koncentrátory a serverem. Oproti UDP protokolu se jedná o poměrně komplikovanou a tedy i časově náročnou komunikaci. Posloupnost komunikace je následující, přičemž na adrese 192.168.0.20 se nachází server:

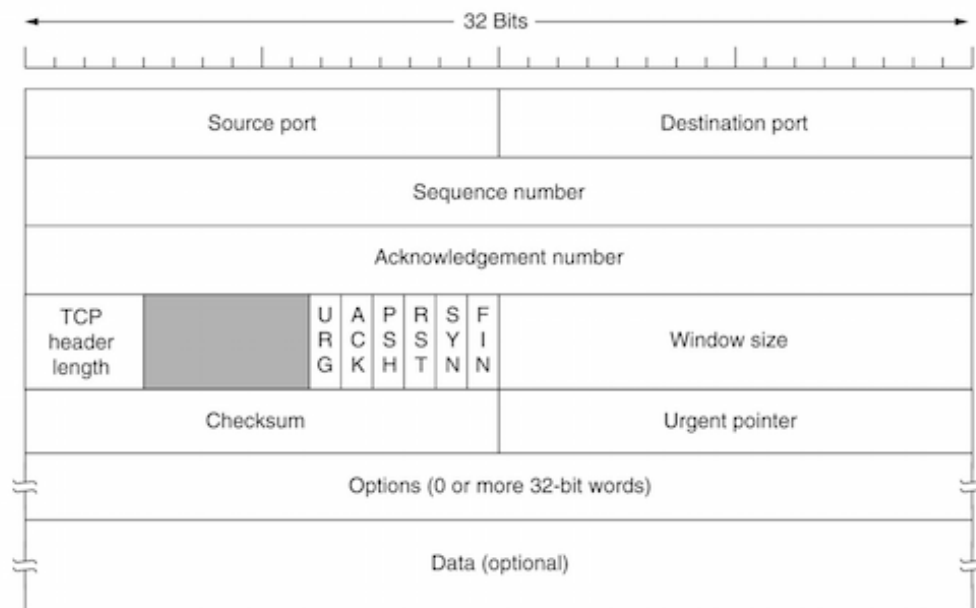
```
1 192.168.0.11 -> 192.168.0.20 : SYN
2 192.168.0.11 <- 192.168.0.20 : SYN, ACK
3 192.168.0.11 -> 192.168.0.20 : ACK
```

Spojení tedy probíhá zhruba následovně. Koncentrátor (192.168.0.11) otevírá TCP spojení vysláním požadavku na synchronizaci příznakem **SYN**. Server potvrzuje spojení pomocí příznaku **ACK** a vysílá také požadavek na synchronizaci (**SYN**). Koncentrátor toto spojení přijímá pomocí **ACK** příznaku, čímž je spojení ustanoveno. Tomuto procesu se říká three-way handshake [3]. Samotné poslání jednoho paketu včetně dat a uzavření spojení pak vypadá následovně:

```
1 192.168.0.11 -> 192.168.0.20 : SYN
2 192.168.0.11 <- 192.168.0.20 : SYN, ACK
3 192.168.0.11 -> 192.168.0.20 : PSH, ACK    # přenos dat
4 192.168.0.11 <- 192.168.0.20 : FIN, ACK
5 192.168.0.11 -> 192.168.0.20 : FIN, ACK
6 192.168.0.11 <- 192.168.0.20 : ACK
```

Začátek spojení (three-way handshake) zůstává stejný, ale pro úsporu množství přenášených informací se hned při potvrzení spojení pomocí **ACK** posílají na server data (**PSH**). Následně server uzavírá spojení a potvrzuje přijetí dat (**FIN, ACK**), koncentrátor uzavírá spojení a potvrzuje uzavření spojení serverem (**FIN, ACK**) a nakonec server potvrzuje uzavření spojení ze strany koncentrátoru (**ACK**). Je tedy zřejmé, že i po **FIN** příznaku dochází k další komunikaci. Nutně tedy tento příznak neznamená úplný konec spojení, ale pouze konec z jedné strany. V tomto případě uzavírá spojení server, ačkoliv spojení nezačínal. Je to vhodné z toho důvodu, že se ušetří jedna cesta pro potvrzení dat na straně serveru a následné ukončení ze strany koncentrátoru. Navíc server by se měl v dané síti chovat velmi zodpovědně, takže nepřipouští otevřená spojení, když k tomu není důvod. V současné chvíli nejsou mezi serverem a koncentrátorem implementovány perzistentní sockety.

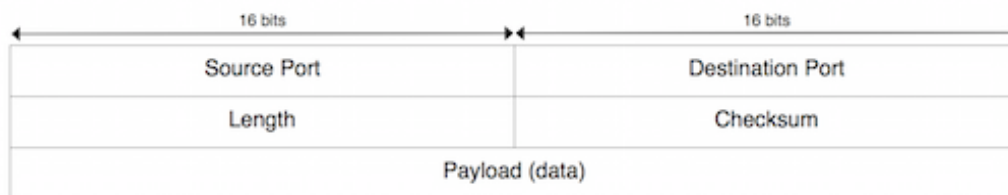
Nespornou výhodou TCP je fakt, že tento protokol zajišťuje to, že daný paket dorazí na cílovou adresu. To například u UDP neplatí. TCP se proto používá pro přenos kritických informací (například vypnutí/zapnutí světla). Jsou to operace, které se musí bezpodmínečně vykonat a jejich nevykonání by vedlo pro uživatele k velmi zvláštnímu chování sítě.



Obrázek 2.1: Uspořádání TCP packetu

2.4 UDP

UDP je oproti TCP protokol typu „fire and forget“. Nestará se tedy o to, jestli informace dorazila na místo určení. To má za následek určitou nespolehlivost přenosu informace, ale o mnohem méně režie potřebné pro přenos. V porovnání s TCP je tento datagram velmi malý:



Obrázek 2.2: Uspořádání UDP datagramu

Pro posílání informace potom stačí vyslat jeden tento datagram s daty a to je vše. Proto skutečnost, že neznáme výsledek přenosu vede k tomu, že musíme být smířeni s faktem, že se některé datagramy jednoduše ztratí např. vlivem kolize s jiným datagramem. Tento problém nastane až u větších sítí, ale při praktickém testování tohoto projektu se ukázalo, že může dojít ke ztrátě datagramů na levnějších komponentách sítě, v tomto případě mohl za ztráty switch. Tento nežádoucí stav může nastat buď vlivem malé paměti daného zařízení, nebo vysokou rychlostí posílání datagramů, což je hlavní příčina. Na levnějších zařízeních pak dochází k přenosu pouze nejrychlejšího náhodného datagramu. Celá síť se pak chová tak, že se datagramy posílají na náhodné prvky za tímto switchem, ale nikdy ne na více než jedno zařízení. Tento efekt byl pozorován na zařízeních ZyXEL ES-105A i TP-LINK TL-WR743ND. U switchu D-Link DFE-916DX k tomuto efektu nedošlo.

Tento přenos (vzhledem k charakteru UDP) je tedy vhodný pro přenos velkého množství informací s tím, že nám případné ztráty nevadí. Typickým zástupcem toho typu přenosu jsou například kontinuální čidla, která neustále snímají (například teplotu) a v krátkých časových intervalech posílají informace na server.

3

Volba vhodné technologie

Pro tuto síť nejsou v tuto chvíli stanoveny zadávající firmou žádné konkrétní požadavky na hardware. Proto je možné vybrat z hlediska softwarového řešení jakoukoliv platformu. Z hardwarového hlediska je doporučeno používat evaluační desky od STMicroelectronics. V následující části budu popisovat jednotlivé použité technologie a důvod jejich volby.

3.1 Prvky senzorické sítě

Prvky senzorické sítě jsou testovány na evaluačních deskách s procesory STM32F207IGH6 a STM32F457IGH6 s využitím oficiálních Cube knihoven [4]. Jedná se o 32-bit procesory pro obecné použití. Pro oba procesory dále platí, že mají 176 pinů s velikostí flash paměti 1024 KB v provedení pouzdra UFBGA pro běžné rozsahy teplot od -40 do 85 °C.

3.1.1 Procesor STM32F207IGH6

- Jádru ARM 32-bit Cortex™-M3 CPU (120 MHz max)
- 128 KB SRAM
- LCD interface
- 12×16 -bit timer, 2×32 -bit timer - až 120 MHz
- 15 komunikačních rozhraní
- $2 \times$ USB, 10/100 Ethernet
- 8 až 14-bit interface pro kameru
- CRC jednotka

3.1.2 Procesor STM32F457IGH6

- Jádru ARM 32-bit Cortex™-M4 CPU (168 MHz max)
- 192 KB SRAM
- LCD interface
- 12×16 -bit timer, 2×32 -bit timer - až 168 MHz
- 15 komunikačních rozhraní
- $2 \times$ USB, 10/100 Ethernet
- 8 až 14-bit interface pro kameru
- CRC jednotka

Veškeré evaluační desky s procesory jsou připojeny pomocí klasických síťových prvků a UTP kabelů do serveru.

3.2 Real-time server

Jako real-time server, který obsluhuje celou síť i webovou aplikaci je zvolen Node.js [5]. Jedná se o platformu postavenou nad V8 JavaScript Engine od společnosti Google. V8 je engine napsaný v C++, který využívá například prohlížeč Google Chrome jako jádro pro velmi rychlé zpracování javascriptu. Díky tomu je možné využívat téměř všech vlastností javascriptu, zejména pak single-thread asynchronní chování (non-blocking I/O model) a EDA architektury. Jedná se o velmi podobný engine jako je v prohlížeči Google Chrome s tím rozdílem, že Node.js neobsahuje možnost práce s oknem programu, nebo s dokumentem jako takovým, protože zde žádný není. Oproti tomu umožňuje přistupovat k process objektu, který zase není dostupný v prohlížečích. Následující ukázka ukazuje jednoduchý webový server napsaný právě s pomocí Node.js:

```
1 var http = require('http');
2 var PORT = 8000;
3
4 var server = http.createServer(function(request, response) {
5     response.writeHead(200, {'content-type': 'text/html'});
6     response.write("<h1>hello</h1>\n");
7     setTimeout(function() {
8         response.end("<p>world</p>\n");
9     }, 2000);
```

```

10 });
11
12 server.listen(PORT, function() {
13     console.log('Server is listening on port ' + PORT);
14 });

```

Tento server je zároveň vysoce škálovatelný a to hlavně do šíře. Je tak možné vytvořit velký počet vzájemně komunikujících uzlů (node), které spolu komunikují. Tyto uzly jsou však striktně odděleny (i na úrovni paměti) a nemohou se tak přímo ovlivňovat. Tento model je vhodný pro velmi vytížené aplikace, protože je možné potřebný výkon rozložit na velké množství méně výkonných strojů.

3.3 Databázový server

Na pozici databázového serveru byl zvolen Redis [6]. Redis je key-value databáze. Od nejrozšířenějších relačních databází se liší například tím, že je násobně výkonnější a neobsahuje relace. Významným prvkem této databáze je právě vztah klíče a hodnoty, kdy hodnotu je možné ukládat do sedmi datových struktur (string, hash, list, set, sorted set, bitmap, hyperloglog). Velkou rychlost Redisu zajišťuje zejména to, že pracuje s RAM pamětí serveru. Přijaté hodnoty si nejdříve ukládá právě do paměti a následně je podle konfigurace ukládá na disk. Příklad výchozí konfigurace:

```

1 # save <seconds> <changes>
2 save 900 1
3 save 300 10
4 save 60 10000

```

Vždy musí být splněna časová podmínka i množství změn za tento čas. Takové nastavení pak tedy udává, že se po 900 vteřinách (15 minut) uloží změny na disk, pokud byla provedena změna alespoň jednoho klíče, nebo po 300 vteřinách při změně alespoň deseti klíčů, resp. po minutě, došlo-li ke změně alespoň 10 000 klíčů. Již z této konfigurace je zřejmé, že se u Redisu očekává velké vytížení a tato databáze je na to připravena.

3.3.1 Redis klíče

Redis klíče mohou být velké až 512 MB a jsou binary safe, tzn. že validní je jak běžný text, tak prázdný text, ale také binární data, nebo obsahy souborů (do maximální velikosti klíče). Samotným klíčům lze pak také nastavovat životnost. Klíče potom expirují po nastaveném čase.

3.3.2 Datová struktura string

Datová struktura string je nejzákladnější způsob jak je možné uložit data do databáze. Princip je velmi jednoduchý. Každému klíči se přiřadí textová hodnota. V této práci je datová struktura string používána k ukládání a inkrementování celkového počtu přijatých nebo odeslaných zpráv. Ukázka použití počítadla:

```
1 > SET counter 1
2 OK
3 > INCR counter
4 (integer) 2
5 > INCRBY counter 50
6 (integer) 102
7 > INCRBYFLOAT counter 3.0e3
8 "3102"
```

Časová náročnost těchto operací je $O(1)$.

3.3.3 Datová struktura hash

Hash je velmi podobný stringu s tím rozdílem, že je možné ukládat k jednomu klíči více hodnot. Časová komplexita je pak $O(1)$ pro jeden prvek resp. $O(N)$ kde N je počet ukládaných nebo vybíraných prvků. V této práci je datová struktura hash použita právě pro ukládání informací o koncovém zařízení. Konkrétně se o koncových zařízeních uchovává IP adresa, porty TCP a UDP komunikace, čas posledního ohlášení, status aktivity a počet přenesených zpráv.

3.3.4 Datová struktura list

List je v Redisu implementovaný jako spojový seznam. Tato implementace umožňuje vkládat nová data na začátek, nebo na konec spojového seznamu v konstantním čase nezávisle na počtu prvků v seznamu. Časová komplexita je tedy opět $O(1)$. U operací kdy se vkládá prvek dovnitř seznamu je komplexita $O(N)$, kdy N je počet prvků, přes které je nutné iterovat, než se dostaneme k požadovanému umístění. V nejlepším případě může být tedy i zde časová komplexita $O(1)$. List je v tomto projektu použitý pro ukládání historie příchozích dat z koncentrátorů následovně:

```
1 > LPUSH device_uid:data <data>
2 > LTRIM device_uid:data 0 999
```

Výhodné na tomto přístupu je to, že je složitost obou příkazů $O(1)$. Je to dáno tím, že LPUSH umisťuje data do seznamu zleva což není nijak časově náročné a dále komplexita LTRIM funkce je dána počtem prvků, které se touto funkcí odstraní, což je jeden starý. S minimální režii je tak uchováváno posledních 1000 hodnot.

3.3.5 Datová struktura set a sorted set

Sety jsou neseřazené kolekce unikátních stringů. Zde je nutné zdůraznit, že hodnoty zde skutečně nejsou seřazené a vrácený výsledek tak může měnit pořadí. Výhodné je, že je možné do této množiny ukládat data se složitostí $O(N)$, kde N je počet prvků. To samé platí i pro čtení. Tato struktura se hodí pro ukládání relací čímž je možné simulovat relační vztahy mezi objekty. V tomto projektu je proto set používán pro ukládání všech známých zařízení k síti a dále pro ukládání vazeb mezi jednotlivými koncentrátory.

Sorted set je obdoba setu s tím rozdílem, že prvky jsou v množině seřazené podle zvoleného skóre. Tato struktura není v projektu nikde použita a je zde uvedena pouze pro úplnost.

3.3.6 Datová struktura bitmap

Bitmapy umožňují ukládat binární data do databáze. Tento způsob práce s daty je pouze obdobou práce se stringy. Vzhledem k tomu, že se jedná o práci s nejmenší jednotkou informace, je tato struktura velmi úsporná co se týče velikosti a velmi se hodí pro ukládání velkého množství pravdivých resp. nepravdivých informací. Bitmapy jsou limitovány na velikost 512 MB stejně jako klíče. To jinými slovy znamená, že je v jednom klíči možné uchovat informace až o 2^{32} stavech.

Tato struktura není v projektu nikde použita a je zde uvedena pouze pro úplnost.

3.3.7 Datová struktura hyperloglog

Posledním a relativně novým datovým typem je hyperloglog. Jedná se o statistickou datovou strukturu, která se používá zejména pro rychlé určení kvantity unikátních dat s chybou méně než 1%. Tato struktura předchází paměťové náročnosti při počítání množství unikátních dat. V běžném případě je totiž nutné pamatovat si tyto data, aby bylo možné při dalším vstupu unikátní hodnotu započítat, nebo ji ignorovat, protože je již započítána. Redis při své implementaci používá konstantní množství paměti, konkrétně 12 kB v nejhorsím případě.

Tato struktura není v projektu nikde použita a je zde uvedena pouze pro úplnost.

[9]

```
1 $ redis-benchmark -q -n 100000 -d 256
2 PING_INLINE: 212314.23 requests per second
3 PING_BULK: 211416.50 requests per second
4 SET: 131752.31 requests per second
5 GET: 199600.80 requests per second
6 INCR: 213219.61 requests per second
7 LPUSH: 213219.61 requests per second
8 LPOP: 204918.03 requests per second
9 SADD: 214592.28 requests per second
10 SPOP: 212765.95 requests per second
11 LPUSH (needed to benchmark LRANGE): 213675.22 requests per
   ↪ second
12 LRANGE_100 (first 100 elements): 45269.35 requests per second
13 LRANGE_300 (first 300 elements): 15586.04 requests per second
14 LRANGE_500 (first 450 elements): 9325.75 requests per second
15 LRANGE_600 (first 600 elements): 6472.49 requests per second
16 MSET (10 keys): 131578.95 requests per second

1 $ redis-benchmark -q -n 100000 -d 256 -P 16
2 PING_INLINE: 1612903.25 requests per second
3 PING_BULK: 2127659.75 requests per second
4 SET: 1086956.50 requests per second
5 GET: 1351351.38 requests per second
6 INCR: 1219512.12 requests per second
7 LPUSH: 934579.44 requests per second
8 LPOP: 1030927.81 requests per second
9 SADD: 1265822.75 requests per second
10 SPOP: 1562499.88 requests per second
11 LPUSH (needed to benchmark LRANGE): 990099.00 requests per
   ↪ second
12 LRANGE_100 (first 100 elements): 35186.49 requests per second
13 LRANGE_300 (first 300 elements): 8521.52 requests per second
14 LRANGE_500 (first 450 elements): 5236.70 requests per second
15 LRANGE_600 (first 600 elements): 3888.48 requests per second
16 MSET (10 keys): 207468.88 requests per second
```

3.3.8 RESP protokol

Redis databáze komunikuje interně přes TCP v RESP (Redis Serialization Protocol) formátu. RESP používá celkem 5 typů dat. Vždy platí, že první byte je byte určující o jaký formát se jedná:

- + jednoduchý string
- - error
- : integer
- \$ bulk string (binary safe)
- * array

Následuje samotný obsah, nebo dodatečné informace, například o délce a vše je ukončeno pomocí CRLF (`\r\n`). Postupně tedy přenášené informace mohou vypadat například takto:

- `+PONG\r\n`
- `-Error 123\r\n`
- `:54986\r\n`
- `$4\r\nPING\r\n` (první část určuje délku bulk stringu, NULL je pak `$-\r\n`)
- `*2\r\n$3\r\nGET\r\n$3\r\nkey\r\n` (první je délka pole, následuje kombinace předchozích)

Redis server potom přijímá podle bulk stringů obsahující jednotlivé instrukce. Tento protokol je velmi důležitý, protože i koncentrátoři posílají data (přes TCP i UDP) v RESP formátu, je tak možné data posílat přímo do databáze. Tato vlastnost však není využívána, protože je vhodné, aby byl jako prostředník server a například zjišťoval aktivitu koncentrátorů. Každopádně tato možnost zde je a pokud by bylo zapotřebí ukládat data tou nejrychlejší cestou, přímý přístup do databáze je tímto možný a funkční.

3.4 Webová aplikace

4

Struktura programového řešení

4.1 Komunikace koncentrátor - server

```
1  /**
2   * @brief Configurates the network interface
3   * @param None
4   * @retval None
5   */
6  static void Netif_Config(void) {
7      struct ip_addr ipaddr;
8      struct ip_addr netmask;
9      struct ip_addr gw;
10
11     IP4_ADDR(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2,
12             ↪ IP_ADDR3);
13     IP4_ADDR(&netmask, NETMASK_ADDR0, NETMASK_ADDR1 ,
14             ↪ NETMASK_ADDR2, NETMASK_ADDR3);
15     IP4_ADDR(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);
16
17     /* Add the network interface */
18     netif_add(&gnetif, &ipaddr, &netmask, &gw, NULL,
19             ↪ &ethernetif_init, &ethernet_input);
20
21     /* Registers the default network interface */
22     netif_set_default(&gnetif);
23
24     if (netif_is_link_up(&gnetif)) {
25         /* When the netif is fully configured this function must
26         ↪ be called */
27     }
28 }
```

```

23     netif_set_up(&gnetif);
24 } else {
25     /* When the netif link is down this function must be
26        ↪ called */
27     netif_set_down(&gnetif);
28 }
29
30 /* Set the link callback function, this function is called
31    ↪ on change of link status*/
32 netif_set_link_callback(&gnetif, ethernetif_update_config);
33 }

```



```

1  udpSocket.on('message', function (msg, rinfo) {
2      sails.log.verbose(JSON.stringify(msg.toString()));
3      //FIXME: not good!
4      if (result =
5          ↪ msg.toString().match(/\*[0-9]+([\r][\n])(\[0-9]+\1([0-9a-z]+\1)+/i))
6          ↪ { //RESP
7          //redisClient.lpush('TEMP_000001:data', result[3]);
8          //redisClient.ltrim('TEMP_000001:data', 0, 999);
9
10         redisClient.lpush('TEMP_000002:data', result[3]);
11         redisClient.ltrim('TEMP_000002:data', 0, 999);
12     }
13     var message = new Buffer('test');
14     udpSocket.send(message, 0, message.length, rinfo.port,
15         ↪ rinfo.address);
16 }).bind(sails.config.globals.UDP_PORT, function () {
17     sails.log('Starting UDP server (port ' +
18         ↪ sails.config.globals.UDP_PORT + ')...');
19 });

```

4.2 Komunikace server - webová aplikace

5

Praktická aplikace

6

Rozšíření stávajícího řešení

IPv6, Bezdrátová komunikace, Zabezpečení, Další prvky sítě

7

Závěr

Literatura

- [1] *I. Fette, Google Inc., A. Melnikov, Isode Ltd.: The WebSocket Protocol* <https://tools.ietf.org/html/rfc6455>
- [2] *University of Telecommunications, Leipzig, Sebastian Lammermann: Ethernet as a Real-Time Technology* http://www.lammermann.eu/wb/media/documents/real-time_ethernet.pdf
- [3] *Barrie Sosinsky: Mistrovství – počítačové sítě*
- [4] *STMicroelectronics: Getting started with STM32CubeF2 firmware package for STM32F2xx series* http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00111485.pdf
- [5] **Node.js:** <http://nodejs.org/>
- [6] **Redis.io:** <http://redis.io/>
- [7] *STMicroelectronics: Description of STM32F4xx HAL drivers* http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00105879.pdf
- [8] *STMicroelectronics: Reference manual - STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx advanced ARM®-based 32-bit MCUs* http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf
- [9] *Redis.io: How fast is Redis* <http://redis.io/topics/benchmarks>