

zcu.png

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

img/zadani1.jpg

img/zadani2.jpg

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	v
Seznam symbolů a zkratk	vi
1 Úvod	1
2 Real-time komunikace	2
2.1 Hardwarové prostředky senzorické sítě	2
2.2 Real-time ve webových aplikacích	4
2.3 TCP	4
2.4 UDP	6
3 Volba vhodné technologie	8
3.1 Prvky senzorické sítě	8
3.2 Real-time server	8
3.3 Databázový server	8
3.3.1 RESP protokol	9
3.4 Webová aplikace	10
4 Struktura programového řešení	11
4.1 Komunikace koncentrátor - server	11
4.2 Komunikace server - webová aplikace	12
5 Praktická aplikace	13
6 Rozšíření stávajícího řešení	14
7 Závěr	15

Seznam obrázků

2.1	Uspořádání TCP packetu	6
2.2	Uspořádání UDP datagramu	6

Seznam symbolů a zkratek

AJAX	Asynchronous JavaScript and XML
AJAJ	Asynchronous JavaScript and JSON
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
RESP	Redis Serialization Protocol
RFC	Request for Comments
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1

Úvod

Cílem této práce je navrhnout real-time komunikaci pro senzorickou síť s přihlédnutím k tomu, že by tato síť měla být ovladatelná 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 provádí časové korekce (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 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 nízkourovňově, následně je propojit s řídicím serverem, na kterém poběží Node.js real-time server 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 rodinného domu resp. jakéhokoliv objektu kde se běžně pohybují lidé a využívají konvenční elektroinstalaci, tzn. například kancelářské budovy, popřípadě jiné objekty podobného charakteru.

2

Real-time komunikace

Real-time komunikace představuje významný prvek v aplikacích, kde je zapotřebí velmi rychlých reakcí systému. Zpravidla se za real-time aplikaci považuje systém, který řeší časové korekce posílaných signálů a tedy vzájemnou časovou synchronizaci vysílače a přijímače. 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. 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 senzorické 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 senzorickou síť potřeba vyřešit tzv. disaster solution, tedy nějaký 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 objektů. 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. AJAJ). 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 [2], které tento systém využívá. Stejně tak jako UDP je zde tento protokol rozbírán zejména z toho důvodu, že právě na TCP packetech 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:

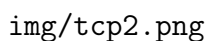
- 1 192.168.0.11 -> 192.168.0.20 : SYN
- 2 192.168.0.11 <- 192.168.0.20 : SYN, ACK
- 3 192.168.0.11 -> 192.168.0.20 : ACK

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 [2]. 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
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 neznamena 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 packet 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 k velmi zvláštnímu chování sítě ze strany uživatele.

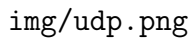


img/tcp2.png

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:



img/udp.png

Obrázek 2.2: Uspořádání UDP datagramu

Pro poslání 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. Tento přenos 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 emitují informace.

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

3.2 Real-time server

3.3 Databázový server

[6]

```
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

```

3.3.1 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 be
26         ↪ called */
27     }
28 }
```

```

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

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

```

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] *Barrie Sosinsky: Mistrovství – počítačové sítě*
- [3] *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
- [4] *STMicroelectronics: Description of STM32F4xx HAL drivers* http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00105879.pdf
- [5] *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
- [6] *Redis.io: How fast is Redis* <http://redis.io/topics/benchmarks>