Tiny Secure Protocol Stack

Jiří Maňák

7. srpna 2021

Obsah

1 Úvod

Před asi dvěma lety jsem poprvé dostal chuť realizovat systém sítě co nejjednodušších senzorů přimykajících se k principu "nainstaluj a zapomeň". Tento nápad se postupně vyvíjel a došel ke koncepci senzorů měřících půdní vlhkost s výdrží baterie přes 5 let.

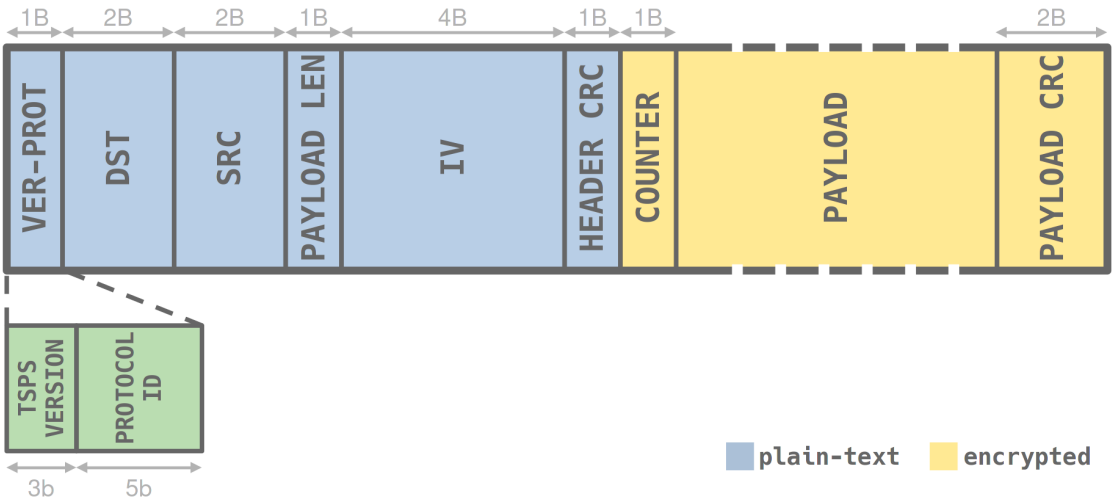
Koncepce se ovšem nestala realitou kromě jednoho návrhu PCB a funkčního dema STM32 s 3.5"kapacitním dotykovým displayem. Nápad ovšem zůstal a opět se vynořil u přednášek předmětu B2B32DATA a začal kulminovat tentokrát z opačného konce - psaním SPI řadiče pro SX1278, SOC v jádru modulu LoRa.

V době psaní této práce projekt přesáhl 15 000 řádků, běží na více než 25 deskách LoRaSTM ve dvou verzích - levnější a jednodušší Node a výkonnější Gateway s USB konektivitou. Má dedikovaný domácí server a webové rozhraní, sbírá enviromentální data, ovládá osvětlení a menší spotřebiče, řídí zalévání.

2 Protokol

TSPS se drží tří hesel: jednoduchost, bezpečnost, modulárnost. Jde o peer-to-peer protokol se zabudovaným šifrováním, který podporuje vrstvení. Je určený pro malé sítě a provozy a má velice nízké nároky na zdroje.

2.1 Formát protokolu



Obrázek 1: Formát hlavičky TSPS

Na obrázku můžeme vidět hlavičku TSPS Core. Většina samotné hlavičky není šifrována. Má oddělený kontrolní součet pro hlavičku samotnou a pro uživatelská data. Obsahuje adresy odesílatele a příjemce, adresy jsou 16 bit, adresa 0 je zakázána. Dále obsahuje Inicializační Vektor potřebný pro dešifrování zbytku paketu. Složitost odpovídá schopnostem protokolu a jeho jednoduchosti

2.2 Šifrování

TSPS používá 128bitů AES v OFB módu, šifrování zprostředkuje oficiální knihovna STM32CryptographicV3.0.0_CM3. Klíče jsou vyměněny a uloženy při párování přes drátové připojení, mohou být kdykoliv opět vyměněny za chodu přes již zabezpečené spojení. Neexistuje podpora pro navázání spojení bez párování fyzickým připojení dvou zařízení.

Pokud TSPS přijme paket a nemá uloženu adresu, paket zahodí. To samé se stane v případě, že po dešifrování nesedí kontrolní součet - tento mechanismus funguje jako autentizace zdroje. Nejsem kryptografickým expertem, ale domnívám se, že tento způsob není považovaný za důvěryhodný, a na lepším autentizačním schematu se pracuje.

Další zajímavostí je zkrácení IV na čtvrtinu což je výsledkem kompromisu mezi užitečnou velikostí paketu a bezpečností přenosu. Vzhledem k relativně nízké frekvenci komunikace těchto zařízení se tento kompromis zdá být adekvátní. Samotné IV je v případě verze Gateway generováno hardwarovým TRNG a v případě Node odvozeno z hodnoty RSSI spektra.

2.3 Párování

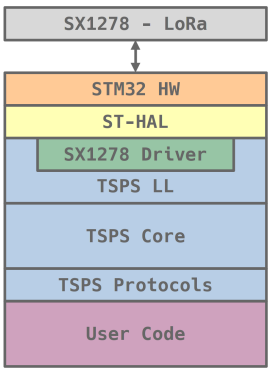
Protokol nemá definovaný způsob na přenos klíčů nedůvěryhodným mediem, spoléhá na párování fyzickým propojením dvou zařízení. Párovací proces zatím řídí počítačová aplikace, nicméně do budoucna se počítá s automatickým párováním hned po spojení, více v sekci Praktické nasazení.

Klíče musí být uloženy v trvalé paměti (v našem případě je to externí EEPROM používána i pro další konfigurační parametry a logging). Klíče jsou v této paměti uloženy bez jakékoliv ochrany, ale tento aspekt je závislý na implementaci. Celkově systém není zabezpečený proti útokům s fyzickým přístupem (i s ohledem na párování, kdy jsou klíče opět přenášeny v plaintextu).

3 Implementace

Při tvorbě návrhu rozdělení softwarových bloků a architektury obecně jsem zvážil několik možných přístupů použitelných v kontextu embedded zařízení - plně asynchronní architektura použitím threadů, interrupt rutiny a nebo zcela blokovací přístup bez současných kontextů. Použití například systému freeRTOS by zvýšilo nároky na zdroje a otevřelo více možností pro chyby implementace, což přesáhlo potenciální benefity. Interrupt rutiny jsem dlouho nechával v rezervě jako ten správný kompromis ale nakonec zůstaly bez využití v TSPS.

Místo toho architektura hojně využívá koncept callback funkcí a má podporu pro plně blokovací uživatelské funkce pro jednoduchost a přehlednost používání. Nevýhodou takového přístupu je nedeterministické chování těchto funkcí, což může být vnímáno v kontextu embedded negativně. Tento problém se dá řešit přesunutím časově kritických bloků právě do periodicky volaných interrupt rutin se kterými TSPS nebude v žádném případě kolidovat protože je nepoužívá.



Obrázek 2: Logická návaznost bloků TSPS (modře) a ostatních FW komponent

3.1 SX1278 Driver

Malá knihovna na obsluhu LoRa modulu, jeho inicializaci, konfiguraci a zápis/výčet bufferů. Používá HAL struktury a funkce. Navazující kód nemusí LoRa modul vnímat jako hromadu registrů.

3.2 TSPS LL

Stavový automat pro obsluhu komunikačního rozhraní. Úkolem je abstrahovat samotný proces příjmu a vysílání paketů pro navazující kód. V našem případě staví na funkcích z SX1278 Driveru, nicméně by to mohlo být i jakékoliv další halfduplexové (bez)drátové rozhraní.

Vždy dá prioritu příjmu proti vysílání, počká na zpracování přijatého paketu po definovaný čas, při překročení tohoto času paket zničí a uvolní rozhraní pro další příjem - nemělo by se stát pokud horní vrstvy fungují korektně. Před odesláním paketu počká krátkou definovanou dobu v řádech milisekund, do budoucna by tento interval měl být náhodný v nějakém rozmezí pro situace, kdy se dvě zařízení opakovaně střetnou při vysílání.

3.3 TSPS Core

Jedná se o hlavní blok s většinou "chytrosti". Definuje funkce pro odesílání dat a systém registrovatelných slotů pro callback funkce náležící TSPS protokolům. Samotné TSPS Core nikdy neodesílá pakety spontánně, pouze přijímá nové zprávy, parsuje, zahazuje nehodící se a zpracovává požadavky z nadřazených bloků.

Pojí se k němu tři hlavní struktury:

```
typedef struct
{
    data_t* data;
    TSPS_Addr_t address;
    uint8_t protocol;
    TSPS_ID_t pcs_id,
        transaction_id;
} TSPS_PCS_t;
```

- Packet Control Struct - struktura popisující příchozí nebo odchozí paket. Obsahuje samotná data paketu, adresu a číslo protokolu
- Je argumentem TSPS Core funkcí sloužících k odesání a callback funkcí protokolů, kde naopak nese přijatá data
- Obě ID zatím nemají využití ale jsou zamýšleny pro složitější protokoly které budou spravovat více transakcí současně

```
typedef struct
{
    bool active;
    TSPS_Tick_t
        initiated,
        timeout;
    TSPS_PCS_t pcs;
} TSPS_Transaction_t;
```

- TSPS Core udržuje seznam aktivních transakcí
- Realizuje obsluhu odpovědí na transakce a prošlé transakce voláním callback funkce příslušného protokolu
- Získané informace jako je latence a ztrátovost používá pro interní stastiky a optimalizaci provozu
- PCS zde slouží pouze jako otisk původní PCS a nenese data.

```
typedef struct
{
    bool success;
    TSPS_Code_t code;
    char* message;
    void* data;
} TSPS_Reply_t;
```

- Stavová struktura vracená TSPS Core funkcemi - informuje volajícího funkce o úspěchu nebo chybě jeho požadavku
- Hodnota *success* je jednoduše definovaná a použije se v případě kdy stejně volající nedokáže ovlivnit úspěšnost akce. Naproti tomu hodnota *code* je enum specifikující konkrétní problém dle které se dá rozhodnout přesněji

TSPS Core tvoří statistiku o všech spárovaných peerech a dále definuje funkce které na základě těchto informací poskytují doporučený počet pokusů na kontakt a doporučený timeout pro odpověď. Také se zaznamenává průměrné vytížení TSPS pro rozhodnutí o provedení méně prioritních akcí, například opětovné kontaktování pravděpodobně neaktivních peerů.

Tyto statistiky nejsou rozsáhlé z důvodu omezení paměti, hlavně na méně výkonných deskách. U peerů se momentálně zaznamenává posledních 5 údajů o latenci a 5 údajů o odpovědi (ztrátovost) použitím exponenciálně váženého průměru. Podobně souhrnné vytížení se zaznamenává každých 10s a uchovává se 60 hodnot opět v exponenciálně váženém průměru.

Údaje o latenci jsou stěžejní pro správné nastavení prodlevy mezi prohlášením, že se pokus kontaktu nezdařil. Údaje o ztrátovosti paketů nastaví počet pokusů navázání spojení, u déle nedostupných zařízení se rozhoduje dle aktuálního vytížení. Pokud je vytížení vysoké, pak je možné, že se pokus o kontakt vůbec neuskuteční, obvzlášť když proběhl nedávno.

Do statistik se také zahrnuje poslední příjem a vysílání peeru, aby bylo možné vzít v potaz staré a tudíž nejspíš neplatné údaje. Do 15 minut stáří se lineárně interpoluje mezi naměřenými hodnotami a maximálními možnými hodnotami nastavenými pevně v konfiguraci. Od 15 minut stáří se používají čistě nejhorší možné hodnoty.

3.4 TSPS Protokoly

Protokoly interagují s TSPS Core a plní příkazy uživatele - poskytují mu služby. Zatím nejpoužívanějším protokolem je Register Access Protocol, který je analogií Modbus TCP/IP ve svojí funkčnosti. Dále již funkční protokol je tzv. Ping protokol, který slouží čistě pro získání odpovědi od peeru a je víceméně takovou předlohou a naprostým základem pro další protokoly.

V době psaní je ve vývoji Communication Control Protocol, který v budoucnu zprostředkuje výměnu klíčů a případné změny kódování/pásma/frekvence komunikace. Výměna klíčů se bude zakládat na statistických datech sledovaných TSPS Core a three-way handshake principu pro samotnou výměnu a potvrzení přijetí. Další plánovaný protokol je obdoba TCP protokolu pro transparentní přenos dat delších než je maximální délka paketu.

Právě Register Access Protocol zprostředkuje momentálně nejdůležitější funkce pro praktické užití této třídy zařízení. Obvykle definuje 256 16-bit signed registrů (kromě zařízení s aktivní konfigurací client-only), u kterých je možno nastavit přístupová práva a ukládat do nich data ze senzorů nebo naopak zapisovat řídicí požadavky. Podporuje dva módy - pooling a subscribe.

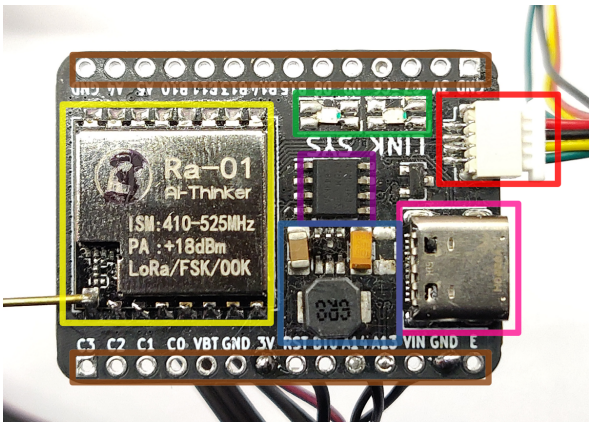
Pooling je jednodušší a hodí se pro zápis nebo výčet dat měnících se v menší míře. Ke každému registru s povolením čtení se může vázat jeden nebo více subscribe adres. Při změně hodnoty registru se automaticky pošle obsah tohoto registru na danou adresu. Adresy je možné přidat nebo odebrat za chodu dalšími peery. Tento přístup je nezbytný pro ostrovní senzory na baterie kde má duty-cycle aktivního stavu zásadní vliv na výdrž baterie nebo pro údaje které je třeba sledovat s nízkou latencí, jako jsou například čidla pohybu.

4 Praktické nasazení

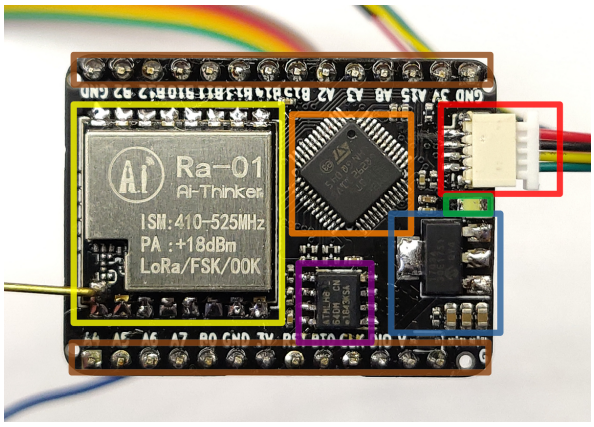
Hlavním tématem práce je protokol samotný, myslím si ale, že nejvěrohodnějším měřítkem úspěšnosti teorie je právě nasazení v praxi. Tato sekce obsahuje shrnutí většiny ostatních prvků potřebných k zavedení tohoto systému do reálná aplikace "chytrého domu".

4.1 LoRaSTM Hardware

Na těchto deskách, podobajících se vývojovým kitům s mikrokontrolery, běží kód popsaný v sekci Implementace. Rozlišení "Node"a "Gateway"je čistě otázka pojmenování, obě zařízení, ač jsou docela rozdílná ve svých schopnostech a výkonu, sdílí 80% kódu, tudíž jejich role nejsou v tomto ohledu definovány a obě mohou být teoreticky použity pro oba účely "Koncového prvku"a "Brány".



(a) **Gateway**: modul LoRa (žlutá), paměť EEPROM (fialová), párovací konektor (červená), zdrojová část (modrá), LED indikátory (zelená), expanzní hřebínky (hnědá), USB-C konektor (růžová), STM32 procesor je na opačné straně



(b) **Node**: modul LoRa (žlutá), paměť EEPROM (fialová), párovací konektor (červená), zdrojová část (modrá), LED indikátor (zelená), expanzní hřebínky (hnědá), STM32 procesor (oranžová)

Obrázek 3: Porovnání rozvržení PCB LoRaSTM a znázornění jejich hlavních komponent

4.1.1 Gateway

Gateway byla první verze a tak i platforma po většinu doby vývoje. Má velkou paměťovou i výkonnostní rezervu díky STM32F217VET. Zároveň má mnoho periferií které nejsou využity, ale mohou se uplatnit v dalších rozšířeních, jako je například DAC a rozhraní pro SD karty.

Díky těmto vlastnostem a svému USB rozhraní je ideální pro funkci Brány k dalším LoRaSTM pro počítač. Procesor není optimalizovaný na nízký odběr, tudíž se nehodí pro dlouhodobé napájení z baterie, navíc spínaný zdroj na desce má vysoký klidový odběr. Další velkou nevýhodou je relativně vysoká cena.

4.1.2 Node

Nová verze Node řeší problémy, které brání Gateway v nasazení ve větším počtu jako báze pro senzory a ostrovní zařízení, přičemž ale dělá kompromisy hlavně ve smyslu dostupné paměti. Její výroba byla navíc komplikována polovodičovou krizí.

V jádru Node je STM32L051C8 procesor řady L hodící se pro bateriové aplikace spolu s jednodušším a mnohem úspornějším zdrojem. Odběr celé desky ve spánku se pohybuje kolem

3 μA, což s periodou aktivity 1 minuta znamená výdrž přes 2 roky na standardním 18650 Li-ion článku a s periodou 15 minut (hodící se pro venkovní teplotní čidlo například) začne být omezujícím faktorem životnost baterie samotné.

Momentálně se využití jak FLASH tak i RAM pohybuje nad 95%, takže jakékoliv další rozšíření se neobejde bez důkladné optimalizace. Po stabilizaci trhu s mikročipy je v plánu další verze Node s dvakrát větší kapacitou obou pamětí.

4.2 Gateway Server

Místo implementace komplikovaných řídicích postupů přímo do mikrokontroleru se o vyčítaní hodnot a kladení požadavků stará Linux server, který zároveň hostuje uživatelsky přístupnou webovou stránku na lokální síti.

Server abstrahuje registry dostupné na různých zařízeních s různými funkcemi do jednotného systému adresovaného jmény místo čísel. Rozlišuje původ registrů a dokáže spravovat registry z jiných zařízení než LoRaSTM nebo dokonce virtuální registry. Zároveň podporuje rozšíření ve formě pluginů napsaných v Pythonu.

Tímto se celá správa přesouvá do ještě vyšších vrstev abstrakce kdy koncové uživatelské aplikace nejsou ovlivněny změnách v hardware nebo rozšíření. Příklad registru tak, jak jej popisuje server, ve formátu JSON:


```
{
    "value": 100,
    "raw_value": 100,
    "updates": [
        {"by": "user", "at": 1626957841, "exp": 0},
        {"by": "user", "at": 1626957826, "exp": 0}
    ],
    "origin": {
        "name": "LoRaSTM",
        "address": 3
    },
    "index": "201",
    "name": "testreg_g",
    "units": "",
    "scale": 1,
    "offset": 0,
    "desc_en": "Set value of the onboard RGB-green led"
}
```