Zabezpečení komunikace chytrého senzoru protokolem TLS na platformě Micropython

Martin Vitoušek*[1], Adam Peichl[1]

[1]*ČVUT v Praze, Fakulta strojní, Ústav přístrojové a řídicí techniky, Technická 4, 166 07 Praha 6, Česká republika*

Abstrakt

Tato práce se zabývá realizací zabezpečení bezdrátové komunikace Wi-Fi s využitím platformy MicroPython. Cílem práce je vytvoření submodulu pro projekt firmy Kyvit s.r.o., jehož myšlenkou je vývoj systému chytrého sensoru (Wisensor) především pro domácí použití. Tento konkrétní submodul má zajistit bezpečnou komunikaci v rámci lokální Wi-Fi sítě mezi serverovým počítačem a mikrokontrolery ESP32, které zajišťují řízení vlastního senzoru. Pro zabezpečení komunikace se v této práci využívá MicroPython knihovna *ussl*, která je odvozena od knihovny *ssl*. S využitím knihovny *ussl* je zajištěno zabezpečení komunikace protokolem TLS (Transport Layer Security).

Klíčová slova: Python, MicroPython, ESP32, TLS, Wi-Fi

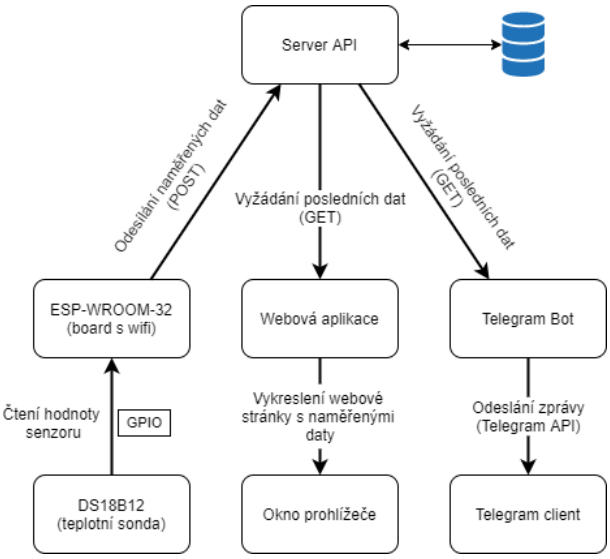
1. Úvod

Hlavním tématem této práce je využití platformy Micropython při realizaci zabezpečení bezdrátové komunikace systému Wisensor[1] (viz. 1.1. Projekt Wisensor) v rámci lokální Wi-Fi sítě. Pro účely práce a vývoj potřebného submodulu společnost Kyvit s.r.o zapůjčila prototyp systému Wisensor. Proto řešení problému bezpečné komunikace musí respektovat restrikce, které přináší celková struktura projektu Wisensor. Především se jedná o omezený výpočetní výkon a další hardwarová omezení malých mikrokontrolerů a současně menší rozsah knihoven platformy MicroPython oproti standardnímu Pythonu.

1.1. Projekt Wisensor

Projekt Wisensor je snaha o vytvoření chytrého modulárního senzoru pro využití v chytrých domácnostech. Celý systém je vyvíjen pomocí jazyka Python, což umožňuje relativně snadný, rychlý a jednoduchý vývoj. V současné době je projekt ve fázi prototypového vývoje, jenž si klade za cíl ověření splnitelnosti všech požadavků s využitím jazyka Python a platformy Micropython. V rámci tohoto vývoje již bylo navrženo a implementováno serverové API, SQL databáze pro ukládání naměřených dat, výstupní webová aplikace, automatický chat bot pro interakci skrze messenger Telegram a základní realizace softwaru pro ovládání samotného senzoru, která zajišťuje chod měření požadovaných veličin a odesílání dat na server s využitím zmíněného API. Obecné zjednodušené schéma systému je vidět na obrázku č.1.

V rámci frontendu je v současné době implementována základní funkcionalita, která uživateli umožňuje nahlížet na automaticky generované výstupy, které shrnují potřebné informace o systému. Tyto náhledy jsou dostupné skrze dvě aktuálně podporovaná rozhraní, tj. webová aplikace a messenger Telegram. Samotný frontend bude dále rozvíjen v rámci dalších iterací v momentě, kdy bude vyžadována možnost řídit akční členy, které budou postupně do systému zahrnuty. Dále je do budoucna plánován vývoj vlastního hardwaru pro systém Wisensor a obecné rozšíření jak v rámci hardwarových komponent, tak i softwarové funkcionality.



Obr. 1. *Struktura projektu*

Pro ověření funkčnosti návrhu a ujasnění základní struktury bylo v rámci prototypového vývoje rozhodnuto o využití ESP32 mikrokontroleru a sondy DS18B20 pro měření teploty. S využitím těchto dvou komponent je možné snadno částečně simulovat samotné měření a komunikaci mezi senzorem a serverem. Samotná komunikace pak probíhá tak, že data získaná teplotní sondou jsou mikrokontrolerem označena příslušným identifikátorem a pomocí HTTP metody POST předána serveru, kde jsou následně zpracována, doplněna o časové razítko a uložena do SQL databáze pro další využití.

[1]Wisensor = pracovní název pro projekt

*Kontakt na autora: Martin.Vitousek@kyvit.cz

1.2. Zabezpečení komunikace - motivace

Zabezpečení komunikace mezi serverem, webovou aplikací a aplikací Telegramového bota není nutné řešit, neboť všechny tyto části v současné chvíli operují v rámci lokální sítě a díky tomu jsou efektivně chráněny od vnějšího útoků. Naproti tomu ESP32 komunikuje se serverem po Wi-Fi síti bez jakékoliv ochrany a tudíž je možné jejich komunikaci snadno odposlechnout nebo i narušit. Narušení komunikace by se možná někomu mohlo zdát jako banální problém v případě, kdy je komunikovanou zprávou pouze údaj o teplotě a ID měření, nicméně v závislosti na dalších okolnostech může tato situace představovat významné riziko.

V nejlepším případě dojde k pouhému znehodnocení změřených dat. Tato situace sice nepředstavuje přímé nebezpečí pro osoby a předměty ve fyzickém světě, nicméně může například zmařit data dlouhodobého automatizovaného měření, na jehož výsledcích má být založena další práce. Naproti tomu poměrně zásadní problém může nastat v případě, kdy bude na základě změřených dat řízen nějaký akční člen. Jako modelovou situaci je možné uvažovat například jednoduchou regulaci výkonu domácího kotle. V takovém případě by mohl případný útočník podvrhnout data a například v zimních mrazech předávat systému informace o tom, že teplota v domě je dostatečná. Tím by donutil kotel snižovat výkon a efektivně by tak odstavil vytápění celého objektu. V případě připojení určitých koncových akčních členů je pak možné, že by potenciální úspěšný útok na systém mohl přímo ohrožovat lidské zdraví.

Výše zmíněné situace představují nepřípustná rizika, která musí být eliminována předtím, než bude možné systém Wisensor bezpečně používat.

2. Analýza

V rámci této části bude detailněji popsána technická realizace projektu ve stavu před zahájením této iterace. Následně bude rozebrána analýza možností pro řešení zabezpečení našeho projektu s ohledem na všechna omezení plynoucí ze struktury celého systému a již zvolených technologií.

2.1. Výchozí situace

Původní řešení, které se nezabývalo bezpečností komunikace, využívalo k přenosu dat, mezi ESP32 a počítačem TCP protokol. Samotná data pak byla posílána jako nezašifrována.

Pro komunikaci se senzorem bylo také vyvinuto vlastní serverové API (Application Programming Interface), které mikrokontroleru umožnilo snadný přístup k hlavní aplikaci. Pro vývoj API je použita knihovna *flask* pro tvorbu webové aplikace [1].

Proces měření a výměna dat probíhal podle následujícího předpisu. Na základě předem určeného nastavení v microkontroleru ESP32 se při rebootu stanovila frekvence snímání hodnot na teplotním senzoru připojenému k desce. Na konci každé periody měření, byla odečtena data ze senzoru a následně proběhla korekce jejich hodnot podle kalibračních konstant. Takto získaná data byla nakonec převedena do formátu JSON a pomocí knihovny *requests* metodou POST odeslána na server k dalšímu zpracování.

Díky požadavku z ESP32 byla na straně serveru zavolána příslušná metoda API rozhraní a požadavek obsloužen. Příchozí data byla vyextrahována a následně jim bylo přiděleno časové razítko a vlastní identifikátor. Poté byl takto vytvořený záznam uložen do databáze pro případné další použití.

Výše popsané řešení sice fungovalo, nicméně komunikace nebyla zabezpečena. Pro demonstraci může posloužit obrázek č.2. Na zmíněném obrázku je zachycen snímek obrazovky programu *Wire Shark*, díky kterému bylo možné odposlouchávat data přenášená mezi ESP32 a počítačem v rámci Wi-Fi sítě. Ve vyznačené oblasti obrázku č.2 je možné rozpoznat data z posílané zprávy. Jednotlivé údaje jsou přenášeny jako dvojice klíč a hodnota. V ukázce jsou vidět tři údaje. První je údaj o teplotě (temperature) s hodnotou „24.5“ (odpovídá °C), následuje tlak (pressure) s hodnotou „*null*“ a nakonec pořadové číslo měření (measurement) s hodnotou *6*.

Hodnota „*null*“ u tlaku není chyba, ale záměr. V dřívější iteraci byl použit senzor BMP280, umožňující měření teploty a tlaku, a tedy hodnota tlaku byla rovněž posílána. Naproti tomu v aktuální verzi je využíván senzor DS18B20. Tento senzor sice měří pouze teplotu okolí, ale jeho výhodu je možnost připojení ke klasickým digitálním GPIO pinům na desce místo využívání komunikace přes I2C. V současné situaci tedy není možné měřit tlak, ale pro demonstraci funkčnosti to není nutné. V případě budoucí potřeby měření tlaku je možné opětovně připojit tlakový senzor a jím naměřená data propagovat skrze výše popsané rozhraní.

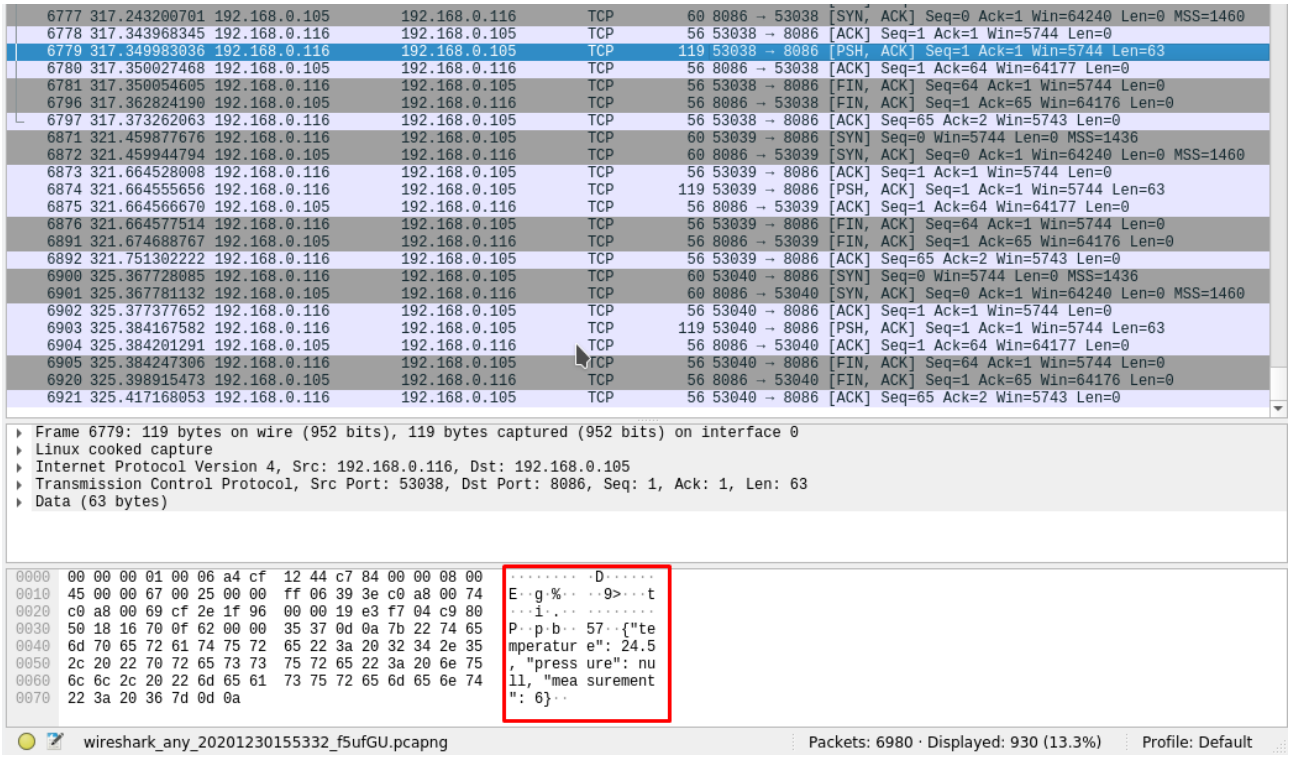
2.2. Možnosti řešení

Při výběru možností pro zabezpečení komunikace bylo nutné respektovat především tato omezení. Prvotně muselo být řešení aplikovatelné s využitím platformy Micropython. Dále pak bylo nutné respektovat omezený výkon mikroprocesoru ESP32 na vývojové desce. Nakonec bylo potřebné přizpůsobit se aktuálnímu stavu projektu v maximální možné míře.

V rámci krátké analýzy bylo zjištěno, že existuje prakticky jediné řešení, které by bylo použitelné pro board ESP32, současně bylo dostupné na platformě Micropython a splňovalo naše požadavky na nenáročnost. V tomto kontextu uvažujeme nenáročnost jak z pohledu implementace, tak i z pohledu HW nároků. Během předchozího vývoje byl odhalen problém se značným přehříváním ESP32. Tento nežádoucí jev je možné omezit snížením „sample rate“ a nebo nižším počtem vykonávaných instrukcí. Z toho důvodu jsme hledali co nejednoduší implementaci zabezpečení, aby nebylo nutné omezovat četnost měření. Ukázalo se, že pro naše účely nejlépe poslouží Micropython knihovna *ussl*[2], která je odvozena od knihovny *ssl* dostupné pro Python. SSL je zkratka pro Secure Sockets Layer, což v doslovném překladu znamená „vrstva bezpečných socketů“ a běžně se používala pro zabezpečení transportní vrstvy, než byla nahrazena modernějším protokolem TLS neboli Transport Layer Security. Z dokumentace bylo zjištěno, že knihovny *ssl* a *ussl* umožňují kromě SSL také využití TLS protokolu [2][3].

Využití tohoto řešení implikuje nutnost úprav komunikace mezi počítačem a ESP32 z předchozí iterace, neboť původní implementace je realizována s pomocí knihovny *requests*. Naproti tomu jakákoliv jiná

[2]Knihovny MicroPythonu odvozené od knihoven Pythonu používají stejné jméno ale s předponou „u“. Písmeno „u“ zde zastupuje řecký symbol „µ“ jako Micro z názvu platformy.



Obr. 2. *Nezabezpečený přenos*

cesta by znamenala implementaci vlastního protokolu pro bezpečnou komunikaci s ohledem na HW možnosti ESP32. Z logických důvodů tedy bylo zvoleno řešení pomocí knihoven *ssl* a *ussl* i za cenu přepracování části původního návrhu.

3. Návrh

V této sekci navážeme na část 2. Analýza a bude zde popsán postup návrhu řešení v závislosti na zjištěných omezeních a současné struktuře projektu. Konkrétně budou přiblíženy možnosti použitých knihoven pro účely řešení našeho problému řešení našeho problému.

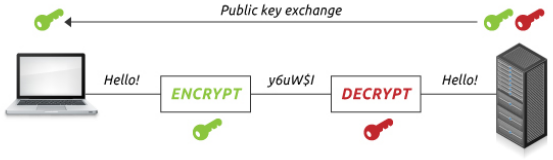
3.1. Knihovna *socket/usocket*

Tato knihovna poskytuje přístup k BSD socketům. BSD sockety jsou souborem funkcí, které umožňují aplikacím přístup k internetové komunikaci. Pro přístup k těmto funkcím slouží specializované API napsané v jazyce C [4].

Samotná Python knihovna *socket* a jí odpovídající knihovna *usocket* pro platformu MicroPython umožňují snadný přístup k API pro BSD sockety. Knihovna umožňuje objektový přístup k rozhraní pomocí funkce *socket()*, která vrací „socketový objekt“. Jednotlivé metody tohoto objektu pak volají konkrétní systémová volání z BSD socket API [2][3].

3.2. Knihovna *ssl/ussl*

Knihovna *ssl* poskytuje nadstavbu nad knihovnou *socket*. Poskytuje přístup k SSL a TLS protokolům pro zabezpečení internetové komunikace. Vzhledem k tomu, že TLS protokol je modernější a obecně lepší než SSL, byl vybrán pro naše řešení. Tato konkrétní knihovna využívá sadu nástrojů OpenSSL [2][3][5].



Obr. 3. *Zjednodušené schéma TLS protokolu; převzato z [6]*

Pro naše účely bylo využito knihovny *ssl* a z ní odvozené knihovny *ussl* pro použití na platformě MicroPython. Tyto moduly poskytují třídu *ssl.SSLSocket*, která nabízí přístup k „socketovému wrapperu“. Wrapper prakticky obaluje socketovou komunikaci a vytváří kolem ní ochranou vrstvu, která na vstupu zašifruje data, následně je bezpečně přenese a na druhé straně je dešifruje. Samotnému poslání dat ještě předchází inicializace komunikace a výměna bezpečnostních klíčů. Zjednodušené schéma je ukázáno na obrázku č.3 [2][3].

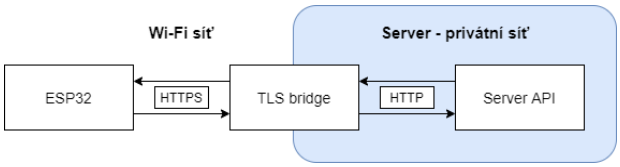
3.3. TLS bridge

Jak již bylo zmíněno v části 2.2. Možnosti řešení současné řešení komunikace mezi vývojovou deskou a počítačem je z pohledu plánovaného rozšíření nedostatečné. Ačkoliv je knihovna *requests* vhodná pro realizaci komunikace mezi ostatními členy a serverovým API, v případě potřeby zabezpečení komunikace na hardwaru mikrokontroleru není použitelná. Na druhou stranu bylo vhodné nějakým způsobem zachovat použití této knihovny pro koncovou část komunikace na straně serveru tak, aby nebyla narušena jednotná struktura fungování serverové aplikace.

Vzhledem k nutnosti využití knihovny *ussl* a s tím spojené knihovny *socket* a současně potřeby zachování knihovny *requests* ze strany serveru, bylo nutné

tyto požadavky zkombinovat. Jako nejjednodušší řešení se nabídlo použití další přidružené aplikace, která byla pracovně označena jako TLS bridge. TLS je „Transport Layer Security“ jak již bylo vysvětleno výše a slovo bridge je samozřejmě v překladu most, nicméně pro naše účely zastupuje význam speciálního mezičlánku potřebného pro některé operace.

Účelem zmíněného „mostu“ je propojení komunikace dvou entit, konkrétně ESP32 a serveru, kde každá entita používá jiný způsob komunikace. Tedy TLS bridge bude na jedné straně přijímat zprávy senzoru v zašifrovaném kanálu pomocí knihovny *ussl*, přijatou zprávu extrahuje a s použitím knihovny *requests* ji propaguje na serverové API, kde je zpráva zpracována stejným způsobem jako tomu bylo doposud. Analogicky bude probíhat komunikace v opačném směru. Server odešle zprávu na TLS bridge, ten ji opět přepracuje a předá ji bezpečně mikrokontroleru. Myšlenka této pomocné aplikace je znázorněna na obrázku č.4.



Obr. 4. *TLS bridge - schéma*

Může se zdát, že vložením tohoto mezičlánku nebyl dříve zmíněný problém s nezabezpečenou komunikací eliminován, ale pouze přesunut mezi dvojici TLS bridge a API, namísto původní dvojice ESP32 a API. Nicméně právě díky tomuto přesunu se nám podařilo nezabezpečený úsek zapouzdřit v rámci privátní sítě serveru (Obr č.4, modrá zóna), kde v současné chvíli předpokládáme, že vzájemná komunikace jednotlivých lokálních aplikací je chráněná.

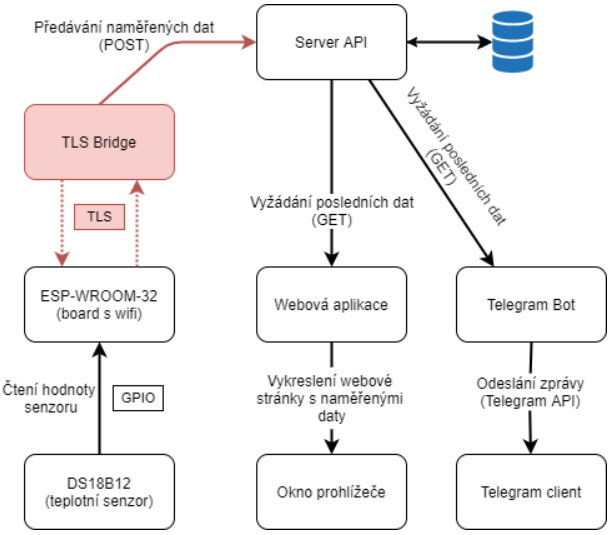
Tímto řešením tedy můžeme uvnitř serveru efektivně schovat otevřenou nezabezpečenou komunikaci a k venkovním periferiím přistupovat výhradně skrze chráněný kanál.

4. Implementace

Stěžejní částí implementace bylo vytvoření výše popsané TLS bridge aplikace a její připojení ke stávajícímu systému. Původní řešení, které je zobrazeno na obrázku č.1, řešilo komunikaci s API pomocí POST metody s využitím knihovny *requests*. Na druhou stranu nově připojený modul musel komunikovat se senzorem pomocí bezpečnostního protokolu TLS.

Z hlediska realizace bylo přistoupeno k řešení co nejjednodušším způsobem. TLS bridge aplikace cyklicky přijímá šifrovaná data od senzoru s pomocí jednotlivých zabezpečených socketů. Následně data rozšifruje a s využitím metody POST knihovny *requests* je předá serveru, kde jsou následně zpracována podle výše popsaného postupu.

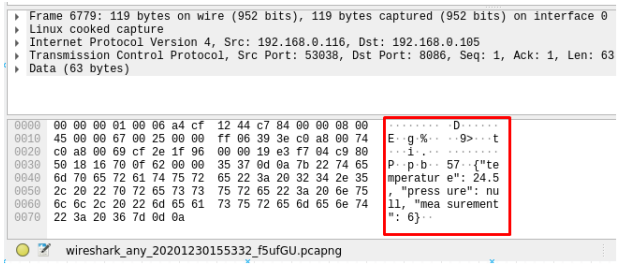
Oproti původní struktuře systému, která byla zobrazena na obrázku č.1, byla připojen nový submodul TLS bridge a spolu s ním bylo implementováno odpovídající komunikační rozhraní. Struktura nového systému je pak zobrazena na obrázku č.5. Rozdílné části schématu byly zdůrazněny pomocí červené barvy.



Obr. 5. *Schéma systému po implementaci změn*

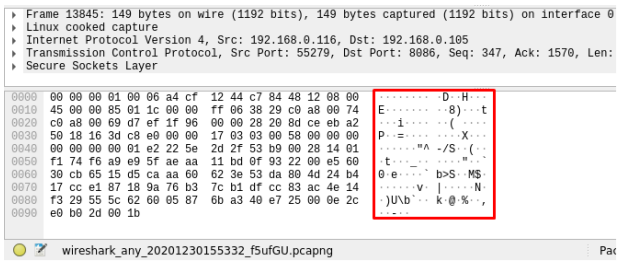
5. Výsledky

Na obrázcích č.6 a č.7 je zobrazeno porovnání zabezpečené a nezabezpečené komunikace. Na obrázku č.6 je ukázán přenos nešifrovaných dat při jejich odposlechnutí pomocí programu Wire Shark, tak jak bylo popsáno v 2.1. Možnosti řešení. Jak je z obrázku patrné, přenášená data jsou viditelná pro vnější subjekty a mohou tak být zneužita.



Obr. 6. *Nezabezpečený přenos*

V druhém případě na obrázku č.7 je zachycena situace po implementaci zabezpečení v podobě TLS protokolu. Analogicky jako u původního řešení jsme se pokusili odposlechnout komunikovanou zprávu. V tomto případě jsme již selhali, neboť jak je vidět na obrázku, data jsou již šifrována a není tedy možné je jednoduše přečíst.



Obr. 7. *Zabezpečený přenos*

6. Závěr

Výsledkem této práce je zabezpečení komunikace systému Wisenzor v lokální Wi-Fi síti, tj. komunikace mezi vývojovou deskou ESP32 a serverovou aplikací.

V rámci řešení tohoto problému bylo nutné nejprve prozkoumat stávající návrh systému a provést analýzu možností řešení. Na základě vyhodnocení struktury systému, HW omezení vyplývajících z využití desky ESP32, dostupnosti knihoven na platformě MicroPython a současně s ohledem na co nejmenší změny bylo navrženo řešení, které využívá knihoven *ssl* a *ussl* pro zabezpečení komunikace pomocí TLS protokolu. Tyto knihovny byly použity pro komunikaci mezi vývojovou deskou a nově vytvořenou aplikací TLS bridge, která umožňuje přístup k privátní síti, kde jsou spouštěny ostatní aplikace systému Wisensor.

Hlavním přínosem této práce (mimo samotné implementace řešení) je ověření realizovatelnosti původního návrhu pouze s využitím prostředků jazyka Python a platformy MicroPython. V návaznosti na výsledky této práce je možné začít řešit přidání řízení koncových akčních členů a dále pokračovat ve vývoji projektu Wisensor.

Poděkování

Tato práce byla podpořena grantem Studentské grantové soutěže ČVUT č. *SGS21/152/OHK2/3T/12*.

Autoři děkují firmě Kyvit s.r.o. za poskytnutí prototypu systému Wisensor.

Literatura

[1] *Flask's documentation*. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/ (cit. 25. 04. 2021).

[2] *Python 3.7.10 documentation*. 2021. Dostupné z: https://docs.python.org/3.7/ (cit. 11. 04. 2021).

[3] *MicroPython documentation*. 2021. Dostupné z: https://docs.micropython.org/en/latest/ (cit. 11. 04. 2021).

[4] *BSD Socket*. Dostupné z: https://www.keil.com/pack/doc/mw6/Network/html/using_network_sockets_bsd.html (cit. 21. 04. 2021).

[5] *OpenSSL*. Dostupné z: https://www.openssl.org (cit. 21. 04. 2021).

[6] *Why is Green Security Padlock Not Showing After Installation of SSL/HTTPS Certificate?* Dostupné z: https://wppathfinder.com/why-is-green-security-padlock-not-showing-after-installation-of-ssl-https-certificate/ (cit. 21. 04. 2021).