VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE A TESTOVÁNÍ PROTOKOLU WIREGUARD

WIREGUARD PROTOCOL IMPLEMENTATION AND TESTING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE **Lukáš Nezbeda**

AUTHOR

VEDOUCÍ PRÁCE **Ing. Ondřej Krajsa, Ph.D.**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Lukáš Nezbeda ***ID:*** 240403

Ročník: 3 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Implementace a testování protokolu Wireguard

POKYNY PRO VYPRACOVÁNÍ:

Cílem této bakalářské práce je implementace protokolu WireGuard na mikrokontroléru ESP32 a následné testování jeho funkčnosti v různých podmínkách. Student na základě dostupné dokumentace a knihoven provede implementaci protokolu WireGuard na mikrokontroléru ESP32. Zaměří se na optimalizaci výkonu a spotřeby energie, aby byl systém vhodný pro nasazení v reálném prostředí. Dále provede testování funkčnosti, přenosové rychlosti a zátěžový test.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 3.6.2025

Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.

prof. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Implementace WireGuard protokolu pro vestavěné systémy, testování vývojové desky ESP32-S3 z pohledu výpočetní a přenosové rychlosti.

KLÍČOVÁ SLOVA

Coremark, ESP-IDF, ESP32, Espressif, Internet of Things, Iperf, lwIP, test přenosové rychlosti, WireGuard, zátěžový test

ABSTRACT

Implementation of the WireGuard protocol for embedded systems, testing the development board ESP32-S3 from the angle of computational power and bandwidth capacity.

KEYWORDS

benchmark, Coremark, ESP-IDF, ESP32, Espressif, Internet of Things, Iperf, lwIP, network troughput test, WireGuard

Vysázeno pomocí balíčku `thesis` verze 4.11; `https://latex.fekt.vut.cz/`

NEZBEDA, Lukáš. *Implementace a testování protokolu Wireguard*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2025. Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Lukáš Nezbeda
VUT ID autora:	240403
Typ práce:	Bakalářská práce
Akademický rok:	2024/25
Téma závěrečné práce:	Implementace a testování protokolu Wireguard

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 závěrečné 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Ondřeji Krajsalovi, Ph.D. za odborné vedení, konzultace, nekonečnou trpělivost a podnětné návrhy k práci.

Obsah

Úvod **21**

1 Řada ESP32 **23**
1.1 Vývojová prostředí 24
1.1.1 Arduino IDE 24
1.1.2 PlatformIO 25
1.1.3 Espressif Iot Development Framework 26
1.2 Iperf 2 29

2 WireGuard **31**
2.1 Wireguard implementace pro lwIP 31
2.2 Rozvržení kódu, a kryptografie 31
2.3 Základní konfigurace 32

3 Řešení implementace **33**
3.1 Výběr programovacího prostředí 33
3.2 WireGuard pro ESP32-S3 33
3.3 Iperf 2 pro ESP32 36
3.4 Monitorování výkonu za běhu 38
3.5 Testování výpočetního výkonu 39

4 Výsledky monitorování **41**
4.1 Analýza paměti 41
4.2 Monitorování paměti za běhu programu 43
4.3 Testování přenosové rychlosti 44

Závěr **47**

Literatura **49**

Seznam symbolů a zkratek **53**

Seznam příloh **55**

Seznam obrázků

1.1 Funkční blokový diagram ESP32 Mikrokontroléru [2] 23
1.2 Arduino IDE 2 25
1.3 PlatformIO správce knihoven 26
1.4 ESP-IDF Menuconfig 27
1.5 Visual Studio Code s ESP-IDF rozšířením 28
1.6 ESP-IDF registr komponentů 28
3.1 Příklad konfigurace WireGuard pro ESP32 34
3.2 WireGuard rozhraní komunikující s ESP32 35
3.3 WireGuard rozhraní komunikující s ESP32 35
3.4 Porovnání komunikace z pohledu Ethernet a WireGuard rozhraní . . 36
3.5 Příklad výstupu komponentu Task_Monitor 38
4.1 Přehled paměti aplikace 41
4.2 Detailní přehled paměti aplikace 42
4.3 Srovnání stavu volné paměti 44
4.4 Porovnání přenosových rychlostí 45

Seznam tabulek

4.1 Seznam velikosti souborů WireGuard knihovny 42
4.2 Volná paměť při šifrovaném provozu 43
4.3 Volná paměť při nešifrovaném provozu 43
4.4 Výsledky průměrů přenosové rychlosti Iperf testů 44

Seznam výpisů

1.1 Výpis Iperf2 testu 29
2.1 Konfigurace WireGuard rozhraní pro Windows zařízení 32
3.1 posloupnost příkazů pro spuštění příkladu pro WireGuard 34
3.2 Funkce iperf_start 37
3.3 Sériový Port ESP32 37
3.4 Výsledek CoreMark testu pro ESP32-S3 39

Úvod

Tato práce se věnuje implementaci WireGuard protokolu pro ESP32 vývojové desky a následné testování funkčnosti, přenosové rychlosti a zátěžového testování. V rámci implementace budou obecně probrány ESP32 mikrokontroléry, vhodná vývojová prostředí, nástroje pro testování a monitorování vývojové desky, WireGuard pro Lightweight IP a jeho implementace pro ESP32. Dále bude sestaven základní tunel mezi mikrokontrolérem a domácím WireGuard serverem pro ověření funkčnosti, sběr a představení poznatků z monitorování.

Kapitola 1 představí krátce ESP32 mikrokontroléry, vývojová prostředí a následně nástroj Iperf. Kapitola 2 uvede WireGuard protokol obecně pro embedded systémy. Kapitola 3 se bude věnovat implementaci veškerých potřebných nástrojů a testování výpočetního výkonu. Na konec kapitola 4 shrne poznatky z monitorování.

1 Řada ESP32

Pro zachování transparentnosti, obsah kapitoly 2, zejména sekce 2.3, 2.1, a 2.2 jsou převzaty z předchozí práce autora této studie a upraveny pro kontext Espressif mikrokontrolérů, autor samostatně přispíval pod dohledem vedoucího. Protože se jedná o nezveřejněnou práci [1]. Citace k původním zdrojům zůstanou zachovány, aby bylo možno dohledat, odkud autor původně čerpal.

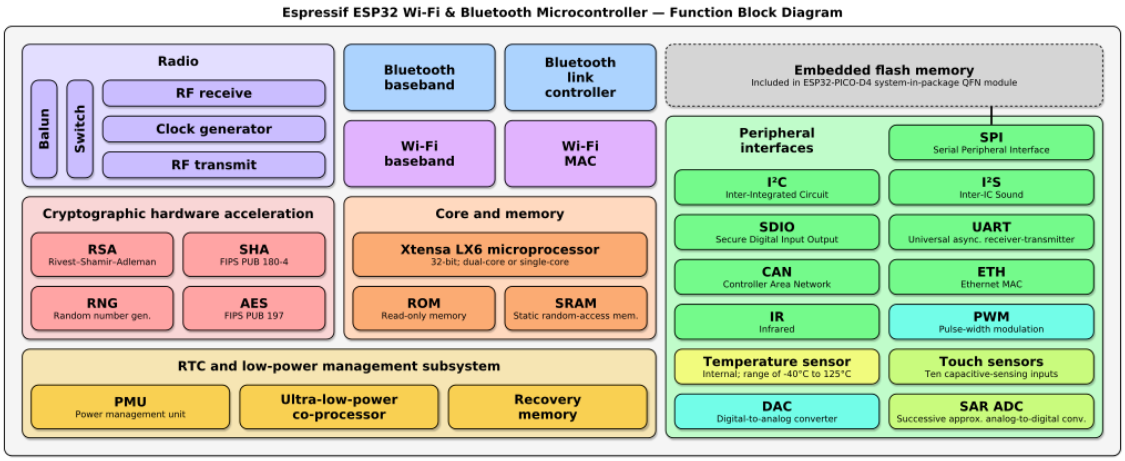
Pro začátek studie povrchově osvětlí ESP32 mikrokontroléry a dostupná programovací prostředí v sekci 1.1. Dále bude věnován čas krátce testovacímu nástroji Iperf2. Pro pokrytí teoretických znalostí pro následný postup implementace.

Vyvinuta společností Expressif Systems, ESP32 je řada SoC mikrokontrolérů s nízkou spotřebou energie a integrovaným Wi-Fi a Bluetooth rozhraním vhodným pro IoT systémy. Jako procesor využívají Tensilica Xtensa LX6 v jednom, či dvoujádrovém provedení, nebo Tensilica Xtensa LX7 také v jednom či dvoujádrovém provedení s frekvencí 240 MHz a pamětí SRAM 520 KB. [2] [3]

Díky mnoha režimům napájení, dynamickému škálování výkonu a funkci pro rychlou změnu taktování lze dosáhnout velmi nízké spotřeby energie [4], s režimem hlubokého spánku odebírajícím 10 μA. [3]

Z bezpečnostní stránky lze také poukázat na Secure boot, šifrování flash paměti, podporu bezpečnostních funkcí standardu IEEE 802.11, a hardwarovou akceleraci pro algoritmy AES, SHA-2, RSA, ECC, a generování náhodných čísel. [5]

Ostatní funkce a celkový rozhled lze pak získat z obrázku 1.1.



Obr. 1.1: Funkční blokový diagram ESP32 Mikrokontroléru [2]

Nejčastěji se lze setkat s ESP32 mikrokontroléry v podobě desky plošných spojů se všemi potřebnými obvody k jeho napájení a programování [6]. Série ESP32 podro-

diny, které se odlišují klíčovými vlastnostmi, příkladem mikroprocesor ESP32-S3 série obsahuje instrukce pro akceleraci práce ve strojovém učení a signálového zpracování. Mikroprocesory C série jsou jednojádrové s instrukční sadou RISC-V. [4] [7]

1.1 Vývojová prostředí

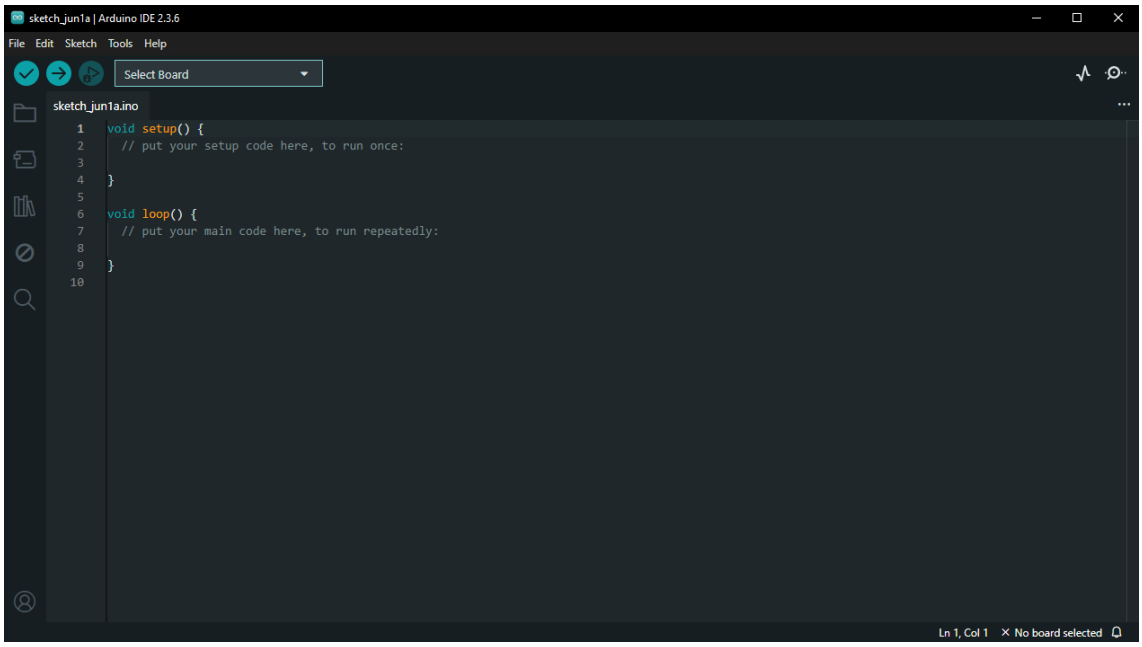
Pro vývoj máme na výběr mnoho možností, které se jednotlivě liší v komplexnosti a vhodnosti dle počátečních programovacích zkušeností. Každé prostředí přináší své výhody a nevýhody, které je potřeba zvážit při tvorbě projektu [8]. Výběr prostředí pro účely studie bude pak rozveden v sekci 3.1

1.1.1 Arduino IDE

Arduino platforma je nejjednodušší způsob, jak se dostat do vývoje pro ESP32 mikrokontroléry. Vývoj programů probíhá v Arduino IDE, kde legacy verze byla napsaná v jazyce Java, a jednalo se o software z výukového prostředí Processing, který byl upraven a rozšířen o funkce jako podpora jazyka Wiring, vytvořený pro programování mikrokontroléru bez specifických znalostí hardwaru a má podobu frameworku v jazyce C++ [8]. Arduino IDE 2 mezitím je z většiny přepracován do TypeScriptu a JavaScriptu, a nesdílí kód s klasickým Arduino IDE. Je založeno na Theila IDE a vytvořeno na platformě Electron [9].

Espressif vytvořil plugin Arduino core for the ESP32, který dodává podporu pro vývoj na mikrokontroléry ESP32 do Arduino IDE prostředí a implementoval podporu FreeRTOS pro programování a exekuci kódu ve formě úloh. Standardně program je rozdělen do funkce `setup`, kde se provádí inicializace hodnot, a funkce `loop`, kde probíhá běh programu, viz 1.2. [8]

Výhodou tohoto prostředí je jednoduchá konfigurace, dobrá komunitní podpora, a velice rychle se dá začít s prototypováním. Arduino IDE je určeno pro jednoduché projekty, složitější projekty rychle ztrácí přehlednost [8] a určité funkcionality dostupné v jiných prostředí nejsou možné bez překompilování knihoven. [10]



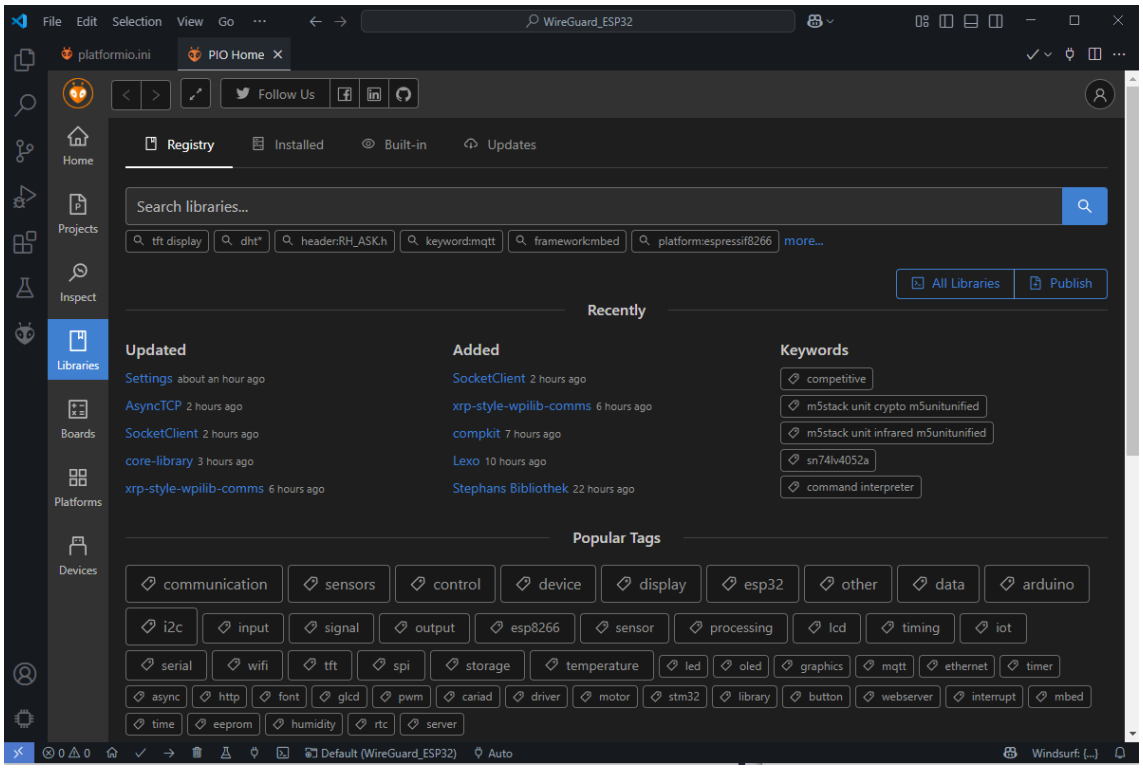
Obr. 1.2: Arduino IDE 2

1.1.2 PlatformIO

I přes popularitu Arduino IDE mezi začátečníky je stále potřeba poukázat na nedostatky, které limitují potenciál projektů, které byly v tomto prostředí vytvořeny, zejména limitace programu do jednoho souboru, globální struktura knihoven a chybějící funkce, bez kterých se programátor dnes neobejde, jako je navigování kódu pomocí referencí, automatické doplňování a integrace s Gitem. [11]

PlatformIO tyto nedostatky doplňuje tím, že umožňuje vývoj embedded systému jako rozšíření ve Visual Studio Code. Práci s projektem lze provádět pomocí příkazového řádku, ale častěji vývojářům pomůže grafické rozhraní pro kompilaci a nahrání programu. PlatformIO sjednocuje desítky vývojových platforem a více než tisíc desek, dále má vlastního správce knihoven, který je vázán jen na určitý projekt, viz 1.3.

I když je PlatformIO trochu složitější pro nováčky než Arduino IDE, stále je možno jej považovat za jedno z přívětivějších prostředí, které udržuje jednoduchost a přináší funkcionalitu zlepšující proces vývoje [11]. Stejně jako v Arduino IDE jsou pro ESP32 určité funkcionality nedostupné, protože PlatformIO na Arduino frameworku pracuje s předkompilovanými knihovnami. [12]



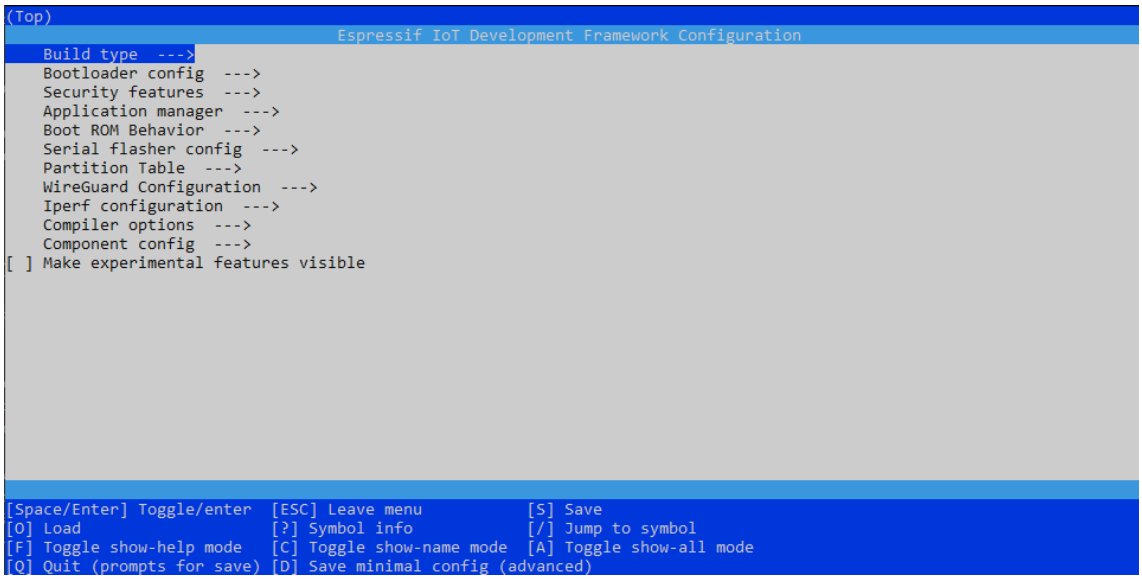
Obr. 1.3: PlatformIO správce knihoven

1.1.3 Espressif Iot Development Framework

Pro složitější a optimalizovanější embedded systémy na základě ESP32 platformy je vhodné využít Espressif IoT Development Framework (ESP-IDF) pro vývoj aplikací nativním způsobem. Tento framework obsahuje možnosti základní konfigurace ESP32 desky, kompilaci zdrojového kódu, a firmware downloader. Pro vývoj aplikací se používá jazyk C. Stejně jako plugin Arduino core for the ESP32, ESP-IDF implementuje podporu pro FreeRTOS. [8]

Vývoj vyžaduje dodržení některých postupů pro úspěšnou kompilaci. Každý projekt vyžaduje soubor `sdkconfig`, který slouží k ukládání konfigurace frameworku pro daný projekt, kde lze nastavit vlastnosti mikrokontroléru, jako zda program využívá FreeRTOS [8], nebo na jaké frekvenci má pracovat. Tyto údaje se generují při kompilaci programu, lze si také vepsat vlastní parametry v případě, že potřebujeme nakonfigurovat dodatečné hodnoty pomocí souboru `Kconfig.projbuild` [13]. Jakékoliv nové soubory zahrnuté v projektu musí být zaznamenány v souboru `CmakeLists.txt`, aby byly při kompilaci nalezeny.

Pro konfiguraci, kompilaci a nahrávání projektu se používá nástroj `idf.py` v příkazovém řádku, který sjednocuje jednotlivé programy využity v procesu prototypování [14]. Obrázek 1.4 zobrazuje příkaz `idf.py menuconfig`, který slouží ke konfiguraci ESP32 desky.

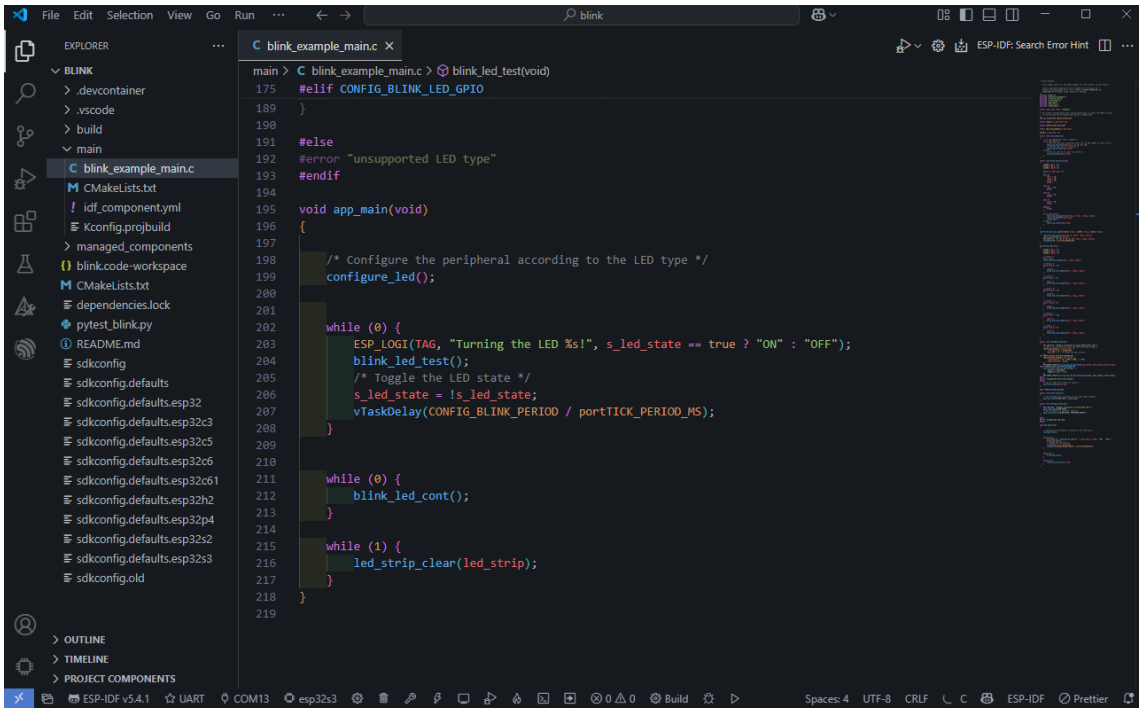


Obr. 1.4: ESP-IDF Menuconfig

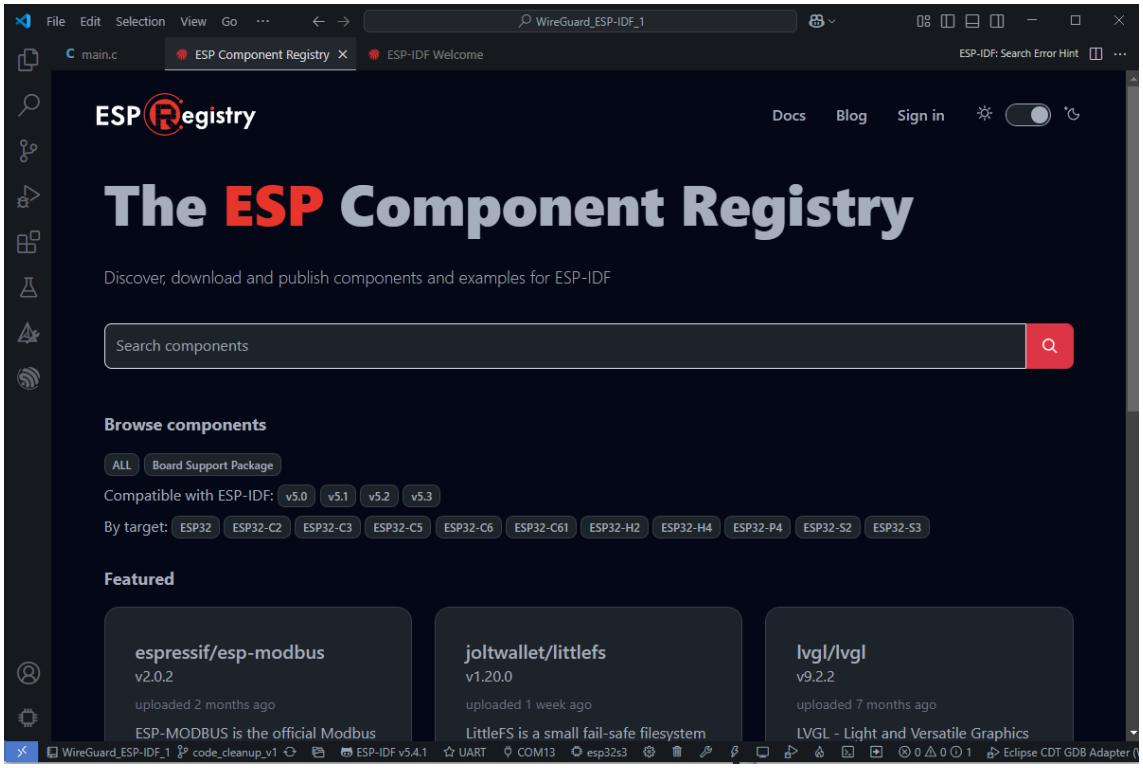
Celková struktura projektu je často kolekce „komponentů“, které slouží jako modulární kusy samostatného kódu, které jsou zkompilovány do statických knihoven a spojeny s aplikací. Mnoho komponentů dodává samotné ESP-IDF, lze ale vytvořit a využít vlastní. [15]

Proto s tímhle frameworkem lze vytvářet aplikace na velmi nízké úrovni, hodí se převážně při tvorbě rozsáhlých a složitých projektů, kde si lze nastavit skoro vše dle potřeb. Nevýhodou pak jsou poměrně vysoké nároky na zkušenosti vývojáře. Komunita není tak rozsáhlá jako v případě Arduino platformy. [8]

Relativně nejjednodušší způsob, jak začít s ESP-IDF, je si jej stáhnout jako rozšíření pro Visual Studio Code, kde mnoho příkazů lze spustit pomocí grafického rozhraní. Konfiguraci projektu lze spravovat přes vestavěný editor na místo `idf.py menuconfig`, a přidávat komponenty pomocí správce IDF komponentů, viz 1.6. Na obrázku 1.5 lze vidět ESP-IDF s částí příkladu pro rozsvícení diody na ESP32 desce.



Obr. 1.5: Visual Studio Code s ESP-IDF rozšířením



Obr. 1.6: ESP-IDF registr komponentů

1.2 Iperf 2

`Iperf2` je testovací nástroj pro měření propustnosti sítě pomocí socketů mezi klientem, který generuje a odesílá data, a serverem, který data přijímá pomocí jak TCP, tak UDP protokolu. Program je spuštěn jako konzolový příkaz a lze jej obohatit argumenty pro preciznější simulaci internetového provozu [16]. Ve výpisu 1.1 lze vidět příklad programu informující o stavu testu. Pro účely studie, Iperf testuje stahování dat na ESP32 mikrokontroléru, který bude fungovat jako Iperf server komunikující s osobním počítačem jako klient, postup bude rozveden v sekci 3.3.

Výpis 1.1: Výpis Iperf2 testu

```
------------------------------------------------------------
Client connecting to 192.168.0.11, TCP port 5201
TCP window size:  125 KByte
------------------------------------------------------------
[  1] local 192.168.0.10 port 59175 connected with
   192.168.0.11 port 5201
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-3.00 sec  1.00 MBytes  2.80 Mbits/sec
[  1] 3.00-6.00 sec  1.00 MBytes  2.80 Mbits/sec
[  1] 6.00-9.00 sec  1.38 MBytes  3.84 Mbits/sec
[  1] 9.00-12.00 sec  1.25 MBytes  3.50 Mbits/sec
[  1] 12.00-15.00 sec  1.25 MBytes  3.50 Mbits/sec
[  1] 15.00-18.00 sec  1.00 MBytes  2.80 Mbits/sec
[  1] 18.00-21.00 sec   896 KBytes  2.45 Mbits/sec
[  1] 21.00-24.00 sec  1.00 MBytes  2.80 Mbits/sec
[  1] 24.00-27.00 sec  1.00 MBytes  2.80 Mbits/sec
[  1] 27.00-30.00 sec  1.00 MBytes  2.80 Mbits/sec
[  1] 0.00-30.03 sec  10.8 MBytes  3.00 Mbits/sec
```

2 WireGuard

V této kapitole práce se pokusí o stručný a krátký popis WireGuard protokolu, a představí ukázku konfigurace rozhraní pro Windows zařízení.

Vytvořen Jasonem Donefeldem, WireGuard je VPN protokol operující na transportní vrstvě s cílem nahradit protokoly IPsec a OpenVPN vyšší bezpečností, výkonem a jednodušším využitím. Tunely fungují na principu asociace mezi veřejným klíčem a zdrojovou IP adresou zařízení. WireGuard využívá dnešní moderní kryptografické funkce jako ChaCha20-Poly1305, řešení pro odolnost vůči denial of service útokům. Protokol se zaměřuje na jednoduchou implementaci a auditování díky své malé velikosti. [17]

2.1 Wireguard implementace pro lwIP

Daniel Hope původně implementoval WireGuard pro Lightweight IP dostupné v repozitáři *wireguard-lwip*. Jeho motivace vycházela z potenciálu protokolu WireGuard jakožto vhodného kandidáta pro vestavěné systémy díky svému základu pro bezpečnost a nízké velikosti kódu. Implementace je navržena tak, aby předešla alokaci paměti přes `malloc()`, držela se pouze jazyka C a Assembly, udržuje nízké nároky na paměť v zásobníku, flash a RAM paměti a kompatibilitu s *lwIP zásobníkem* [18].

2.2 Rozvržení kódu, a kryptografie

wireguard-lwip se dělí na 4 části [18]:

- `wireguard.c`
- `wireguardif.c`
- `wireguard-platform.c`
- *crypto code*

Ve `wireguard.c` leží celková implementace protokolu WireGuard, která není specifická pro IP zásobník. `wireguardif.c` obsahuje integraci pro lwIP, `netif` rozhraní a handling periodických úkonů potřebných pro WireGuard, příkladem časovače, a `keepalive` [18].

Dále `wireguard-platform.h` slouží pro uložení definice funkcí závislých na platformě, při integraci je potřeba tyto funkce implementovat dle platformy [18]:

```
uint32_t wireguard_sys_now();
void wireguard_tai64n_now(uint8_t *output);
bool wireguard_is_under_load();
void wireguard_random_bytes(void *bytes, size_t size);
```

Na konec *Crypto code*, zde se nachází kryptografické funkce, které WireGuard využívá. Tato část má potenciál pro další optimalizaci, jak zmiňuje Daniel Hope v `README.md` repozitáře: „Dodané kryptografické rutiny jsou kompletně napsané v jazyce C a nejsou optimalizovány pro specifickou platformu. Jsou funkční, jejich spotřeba paměti je nízká, ale budou pravděpodobně pomalé na vaší platformě.“ Funkce tedy lze implementovat s optimalizacemi nebo zaměnit za specifickou dostupnou implementaci, která je už optimalizovaná pro danou platformu. [18]

2.3 Základní konfigurace

Pro ukázku konfigurace schopné komunikace v tunelu slouží výpis 2.1, příklad byl inspirován z repozitáře neoficiální komunitní dokumentace pro WireGuard *wireguard-docs* [19], a návodu od Introserv [20]. Tato konfigurace bude sloužit jako ukázka pro server, se kterým bude komunikovat testovaný ESP32 mikrokontrolér, konfigurace pro ESP32 bude rozvedena v sekci 3.2.

Výpis 2.1: Konfigurace WireGuard rozhraní pro Windows zařízení

```
1  # Nastavení serveru
2  [Interface]
3  # Soukromý klíč vygeneruje automaticky WireGuard, z toho
     v rozhraní derivuje veřejný klíč pro ostatní zařízení
4  PrivateKey = MArq1UkZceq/5QpCTSkpdUYkZuy2Lb6JQqFqJGy6l1g=
5  ListenPort = 11010
6  # Rozsah adres ve WireGuard tunelu
7  Address = 10.0.0.1/24
8
9  # Nastavení klienta v tunelu (ESP32)
10 [Peer]
11 # Veřejný klíč klienta
12 PublicKey = sbgiQgldFJXTyJtui2+U3DsXXAsUKkf4V5Zlxm811R0=
13 # Předsdílený klíč
14 PresharedKey =
     wMwdpGBdGQbeJQV1MKgwf8nYDHe42cgzgVjjBHMPOtM=
15 # IP Adresa klienta v tunelu
16 AllowedIPs = 10.0.0.2/32
17 # Reálná adresa klienta s portem
18 Endpoint = 192.168.0.11:11010
```

3 Řešení implementace

Tato kapitola se soustředí na řešení implementace WireGuard pro ESP32 mikrokontroléry a jeho následné testování. Nejdříve shrne cíle a stanoví dostupné zdroje k jejich splnění. Sekce 3.1 vysvětlí postup, při kterém bylo vybráno programovací prostředí k sestavení projektu, který má zobrazit výsledky v kapitole 4, a proč jiné alternativy nemohou vyhovovat požadavkům. Sekce 3.2, 3.3, 3.4 zvolí vhodné komponenty pro sestavení projektu a vysvětlí, jak byly jednotlivě implementovány. Na konec studie provede CoreMark test výpočetního výkonu a zobrazí výsledek.

3.1 Výběr programovacího prostředí

Pro splnění cílů studie je potřeba WireGuard nejen implementovat, ale následně otestovat simulovaným provozem, analyzovat stav paměti při nahrání, v době běhu programu a otestovat výkon samotného mikročipu, a spotřebu v nezatíženém či zatíženém stavu.

Lze určit, že se jedná o projekt nad rámec Arduino IDE, a bylo by moudřejší přejít na sofistikovanější programovací prostředí, jako třeba PlatformIO. Chceme-li sledovat stav za běhu programu, je potřeba zvážit práci s FreeRTOS, hlavně protože má nastavitelné funkcionality pro generování statistik za běhu programu, které budou blíže uvedeny v sekci 3.4, je ale potřeba mít přístup ke konfiguraci `sdkconfig`, což je možné pouze pod Espressif frameworkem, chceme-li pokračovat s Arduino frameworkem, je potřeba překompilovat knihovny, jak bylo zmíněno v sekci 1.3.

Buď lze pracovat v PlatformIO s Espressif frameworkem, nebo rovnou použít Espressif IoT Development Framework, pro účely studie byl zvolen ESP-IDF Visual Studio Code rozšíření jakožto nejvíce nativní volba pro vývoj na ESP32 mikročipu.

3.2 WireGuard pro ESP32-S3

Jako základ projektu byl vybrán demo příklad dostupný z repozitáře *esp_wireguard* [21], který úspěšně vytvoří tunel mezi mikročipem, následně začne posílat `ICMP Echo` na server, odpojí se a zopakuje cyklus. Tento příklad společně s potřebným komponentem lze provést pomocí ESP-IDF terminálu nebo získat z registru, stačí si jej vyhledat, přejít do *examples*, *demo*, a zvolit *Create project from this example*. Následně se vytvoří nový projekt, do kterého je pak potřeba nainstalovat již zmíněný komponent.

Po úspěšné instalaci je potřeba u projektu nastavit cílové zařízení a cílový port, a nakonfigurovat `sdkconfig` pro mikročip, zkompilovat, nahrát a monitorovat. Po-

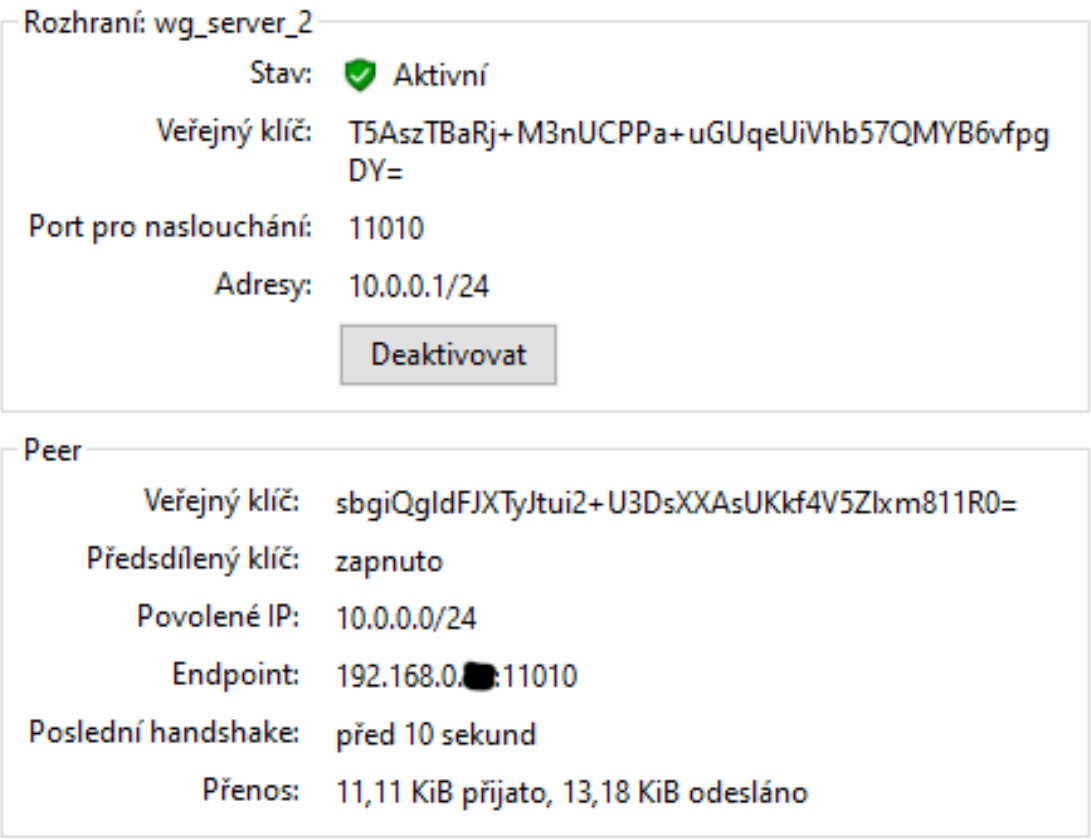
sloupnost těchto příkazů je dostupná ve výpisu 3.1, snímky 3.2 a 3.3 pak zobrazují funkční tunelové připojení mezi ESP čipem a serverem, příklad konfigurace lze vidět ve snímku 3.1. V konfiguraci je důležité nastavit podporu protokolu Point-to-Point.

Výpis 3.1: posloupnost příkazů pro spuštění příkladu pro WireGuard

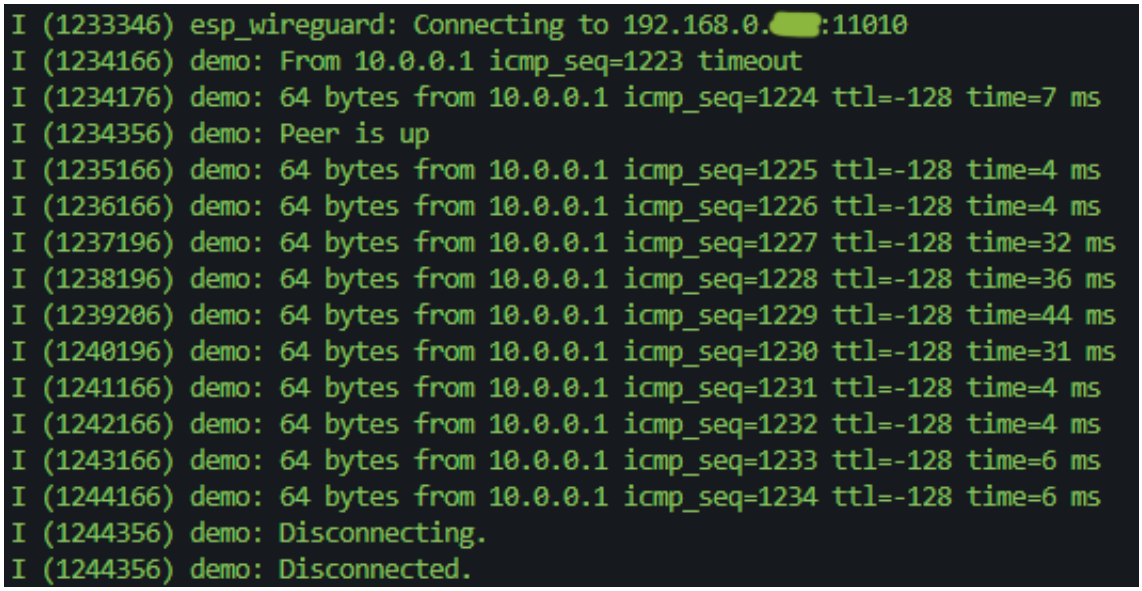
```
1 idf.py create-project-from-example "trombik/esp_wireguard
    =0.9.0:demo"
2 idf.py add-dependency "trombik/esp_wireguard^0.9.0"
3 idf.py set-target <esp-32>
4 idf.py menuconfig
5 idf.py -p <port> build flash
```

```
#
# Example Configuration
#
CONFIG_ESP_WIFI_SSID="SSID"
CONFIG_ESP_WIFI_PASSWORD="PWD"
CONFIG_ESP_MAXIMUM_RETRY=5
CONFIG_WG_PRIVATE_KEY="sHIJdwUHWAxaoKHYTohb3ScSNsc13eANEYJn628veV8="
CONFIG_WG_LOCAL_IP_ADDRESS="10.0.0.2"
CONFIG_WG_LOCAL_IP_NETMASK="255.255.255.0"
CONFIG_WG_LOCAL_PORT=11010
CONFIG_WG_PEER_PUBLIC_KEY="T5AszTBaRj+M3nUCPPa+uGUqeUiVhb57QMYB6vfpgDY="
CONFIG_WG_PRESHARED_KEY="wMwdpGBdGQbeJQV1MKgwf8nYDHe42cgzgVjjBHMPOtM="
CONFIG_WG_PEER_ADDRESS="192.168.0.10"
CONFIG_WG_PEER_PORT=11010
CONFIG_WG_PERSISTENT_KEEP_ALIVE=0
CONFIG_EXAMPLE_PING_ADDRESS="10.0.0.1"
# end of Example Configuration
```

Obr. 3.1: Příklad konfigurace WireGuard pro ESP32



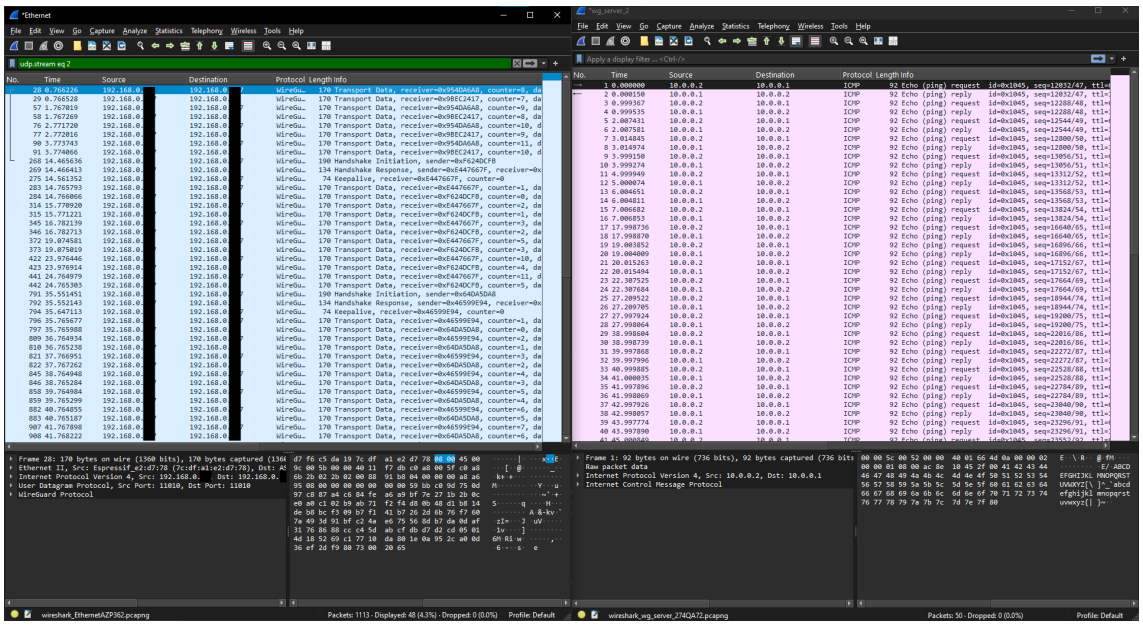
Obr. 3.2: WireGuard rozhraní komunikující s ESP32



```
I (1233346) esp_wireguard: Connecting to 192.168.0.[redacted]:11010
I (1234166) demo: From 10.0.0.1 icmp_seq=1223 timeout
I (1234176) demo: 64 bytes from 10.0.0.1 icmp_seq=1224 ttl=-128 time=7 ms
I (1234356) demo: Peer is up
I (1235166) demo: 64 bytes from 10.0.0.1 icmp_seq=1225 ttl=-128 time=4 ms
I (1236166) demo: 64 bytes from 10.0.0.1 icmp_seq=1226 ttl=-128 time=4 ms
I (1237196) demo: 64 bytes from 10.0.0.1 icmp_seq=1227 ttl=-128 time=32 ms
I (1238196) demo: 64 bytes from 10.0.0.1 icmp_seq=1228 ttl=-128 time=36 ms
I (1239206) demo: 64 bytes from 10.0.0.1 icmp_seq=1229 ttl=-128 time=44 ms
I (1240196) demo: 64 bytes from 10.0.0.1 icmp_seq=1230 ttl=-128 time=31 ms
I (1241166) demo: 64 bytes from 10.0.0.1 icmp_seq=1231 ttl=-128 time=4 ms
I (1242166) demo: 64 bytes from 10.0.0.1 icmp_seq=1232 ttl=-128 time=4 ms
I (1243166) demo: 64 bytes from 10.0.0.1 icmp_seq=1233 ttl=-128 time=6 ms
I (1244166) demo: 64 bytes from 10.0.0.1 icmp_seq=1234 ttl=-128 time=6 ms
I (1244356) demo: Disconnecting.
I (1244356) demo: Disconnected.
```

Obr. 3.3: WireGuard rozhraní komunikující s ESP32

Ověření, zda je provoz šifrovaný, si lze ověřit zachycením paketů v programu Wireshark. Snímek 3.4 ukazuje porovnání komunikace z pohledu ethernetového rozhraní a WireGuard rozhraní.



Obr. 3.4: Porovnání komunikace z pohledu Ethernet a WireGuard rozhraní

Tímhle funkcionalita je dosažena, ICMP zprávy ale nebudou dostatečné pro testování přenosové rychlosti, k tomu lze použít Iperf 2 komponent, více v následující sekci.

3.3 Iperf 2 pro ESP32

Stejně jako WireGuard, Iperf lze přidat pomocí registru či konzole, dostupné v repozitářích *iperf* [22] a *iperf-cmd* [23].

V hlavní složce projektu je pak potřeba následně rozšířit `CMakeLists.txt` o položku `console`, v kategorii `REQUIRES`, aby se program správně zkompiloval.

V případě studie, ESP32 bude fungovat jako Iperf server. Po úpravě projektu program volá funkci `iperf_start (iperf_cfg_t *cfg)`, které je předán ukazatel na konfiguraci, ESP32 otevře socket a začne naslouchat pro připojení. Funkce je zobrazena ve výpisu 3.2.

V konfiguraci určíme hlavně vlastnosti `.interval`, `.time` a `len_send_buf`, kde určíme interval hlášení zpráv, dobu trvání testu a velikost bufferu pro socket. V konfiguraci je potřeba nastavit hodnoty CONFIG_IPERF_SOCKET_RX_TIMEOUT a CONFIG_IPERF_SOCKET_TCP_TX_TIMEOUT pro zamezení samotnému ukončení socketu bez provedení testu.

Výpis 3.2: Funkce iperf_start

```
#include "iperf.h"

static void start_iperf_server()
{
    iperf_cfg_t iperf_cfg = {
        .type           = IPERF_IP_TYPE_IPV4,
        .flag           = IPERF_FLAG_SERVER,
        .source_ip4     = IPERF_SOURCE_IP_ADDRESS,
        .sport          = IPERF_SOURCE_IP_PORT,
        .dport          = IPERF_DESTINATION_IP_PORT,
        .len_send_buf   = IPERF_LEN_SEND_BUFFER,
        .interval       = IPERF_INTERVAL,
        .time           = IPERF_TIME,
    };

    ESP_ERROR_CHECK(iperf_start(&iperf_cfg));
}
```

Jakmile je Iperf spuštěn na ESP32, klient může spustit test a zjistit přenosovou rychlost pro standardní a WireGuard rozhraní, výpis 3.3 zobrazuje příklad výstupu z ESP32.

Výpis 3.3: Sériový Port ESP32

```
I (143577) iperf: accept: 192.168.0.10,63338

Interval       Bandwidth
 0.0- 3.0 sec  2.66 Mbits/sec
 3.0- 6.0 sec  3.92 Mbits/sec
 6.0- 9.0 sec  3.19 Mbits/sec
 9.0-12.0 sec  3.28 Mbits/sec
12.0-15.0 sec  5.20 Mbits/sec
15.0-18.0 sec  4.59 Mbits/sec
18.0-21.0 sec  2.46 Mbits/sec
21.0-24.0 sec  1.79 Mbits/sec
24.0-27.0 sec  2.85 Mbits/sec
27.0-30.0 sec  4.12 Mbits/sec
 0.0-30.0 sec  3.77 Mbits/sec
```

3.4 Monitorování výkonu za běhu

Pro monitorování lze použít *ESP32_Task_Monitor* [24], na rozdíl od ostatních komponentů, tento není v registru dostupný, proto je potřeba jej manuálně naklonovat z Gitu a odkázat na něj v `CMakeLists.txt`. V projektu lze vytvořit složku *components* do které můžeme komponent uložit. V `CMakeLists.txt` pak je potřeba doplnit cestu zdrojového kódu komponentu k `SRCS` a složku hlavičkového souboru komponentu k `INLCLUDE_DIRS`.

Následně je potřeba nakonfigurovat `sdkconfig` a povolit následující funkce [24]:

- `USE_TRACE_FACILITY`
- `USE_STATS_FORMATTING_FUNCTIONS`
- `GENERATE_RUN_TIME_STATS`
- `Enable display of xCoreID in vTaskList`

V hlavním zdrojovém kódu pak lze doplnit pouze funkci `task_monitor()` a hlavičkový soubor `#include "task_monitor.h"`. Výstup projektu lze vidět na snímku 3.5, *ESP32_Task_Monitor* sleduje každou FreeRTOS úlohu, na kterém jádře běží, v jakém je stavu, kolik paměti v bajtech zabírá v zásobníku, dobu běhu v mikrosekundách a poměr mezi celkovým časem, kdy program běží, kdy jednotlivá úloha momentálně pracuje, a celkový čas, kdy program běží od zapnutí, a od inicializace *ESP32_Task_Monitor* úlohy. [24]

Nejpodstatnější část, kterou můžeme sledovat, jsou celková paměť *Total heap size*, momentálně dostupná paměť *Current heap free size* a nejméně volné (nejvíce zabrané) paměti v době běhu programu *Minimum heap free size* [25].

```
I (4076140) tm: TASK NAME:         STATE:     CORE:  NUMBER:  PRIORITY:  STACK_MIN:  RUNTIME, µs:   RUNTIME, %:
I (4076140) tm: IDLE0              Ready      0      5        0          688         3955454130     97.039
I (4076150) tm: monitor_task       Running    0      13       1          3908        74338510       1.824
I (4076160) tm: wifi               Blocked    0      10       23         3468        10982947       0.269
I (4076170) tm: esp_timer          Suspended  0      3        22         3008        755954         0.019
I (4076180) tm: main               Blocked    0      4        1          1192        196074         0.005
I (4076190) tm: ipc0               Suspended  0      1        24         528         32178          0.001
I (4076280) tm: sys_evt            Blocked    0      9        20         584         4529           0.000
I (4076290) tm: IDLE1              Ready      1      6        0          696         4074566604     99.961
I (4076300) tm: tiT                Blocked    Any    8        18         1200        35583706       0.873
I (4076310) tm: iperf_server       Blocked    Any    11       3          2028        154521         0.004
I (4076320) tm: ipc1               Suspended  1      2        24         536         54832          0.001
I (4076330) tm: iperf_traffic      Blocked    1      147      4          1892        728            0.000
I (4076340) tm: Tmr Svc            Blocked    Any    7        1          1312        17             0.000
I (4076360) tm: Total heap size:        349392 bytes
I (4076360) tm: Current heap free size: 238924 bytes (68.38 %)
I (4076370) tm: Minimum heap free size: 231964 bytes (66.39 %)
I (4076380) tm: Total RunTime: 4076168123 µs (4076 seconds)
I (4076390) tm: System UpTime: 4076422380 µs (4076 seconds)
```

Obr. 3.5: Příklad výstupu komponentu Task_Monitor

3.5 Testování výpočetního výkonu

Pro zátěžové testování byla využita CoreMark implementace dostupná v repozitáři [26]. CoreMark, který je známý pro testování embedded systémů s cílem nahradit Dhrystone a napravit jeho nedostatky, [27]. CoreMark není zahrnut do projektu, ale byl načten krátce pro získání skóre bez interference zbytku kódu. Z testu dostupného ve výpisu 3.4 lze vidět, že konečné CoreMark skóre je 611,416820.

Výpis 3.4: Výsledek CoreMark testu pro ESP32-S3

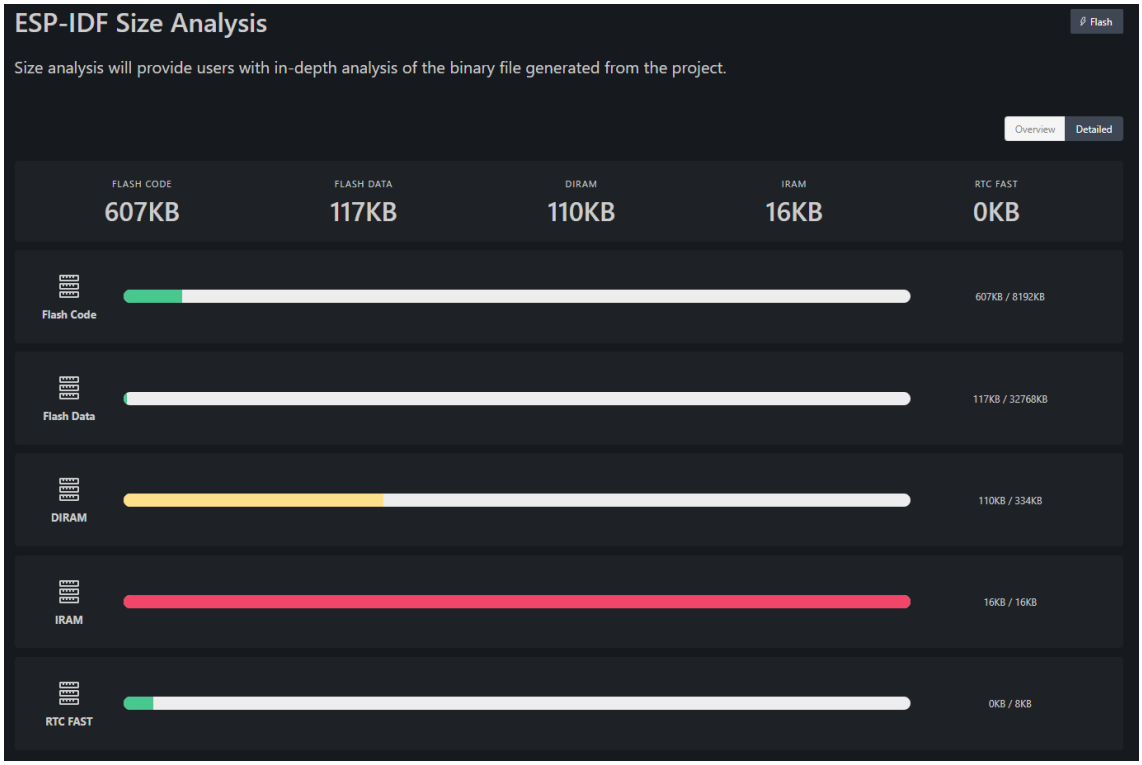
```
1  Running coremark...
2  2K performance run parameters for coremark.
3  CoreMark Size    : 666
4  Total ticks      : 17991
5  Total time (secs): 17.991000
6  Iterations/Sec   : 611.416820
7  Iterations       : 11000
8  Compiler version : GCC14.2.0
9  Compiler flags   : -ffunction-sections -fdata-sections -
     gdwarf-4 -ggdb -Og -fno-shrink-wrap -fstrict-volatile-
     bitfields -fno-jump-tables -fno-tree-switch-conversion
      -std=gnu17 -O3 -fjump-tables -ftree-switch-conversion
10 Memory location  : IRAM
11 seedcrc          : 0xe9f5
12 [0]crclist       : 0xe714
13 [0]crcmatrix     : 0x1fd7
14 [0]crcstate      : 0x8e3a
15 [0]crcfinal      : 0x33ff
16 Correct operation validated. See README.md for run and
     reporting rules.
17 CoreMark 1.0 : 611.416820 / GCC14.2.0 -ffunction-sections
      -fdata-sections -gdwarf-4 -ggdb -Og -fno-shrink-wrap
     -fstrict-volatile-bitfields -fno-jump-tables -fno-tree
     -switch-conversion -std=gnu17 -O3 -fjump-tables -ftree
     -switch-conversion / IRAM
18 CPU frequency: 240 MHz
```

4 Výsledky monitorování

V této kapitole práce zobrazí výsledky testů z implementace a měření protokolu, a vysvětlí jak byly hodnoty z testů získány. Sekce 4.1 projde výsledky analýzy paměti a podrobně se zaměří pouze na WireGuard, v sekci 4.2 studie zobrazí statistiky alokované dynamické paměti za běhu aplikace, v sekci 4.3 zobrazí výsledky testů zašifrovaného a nezašifrovaného síťového provozu.

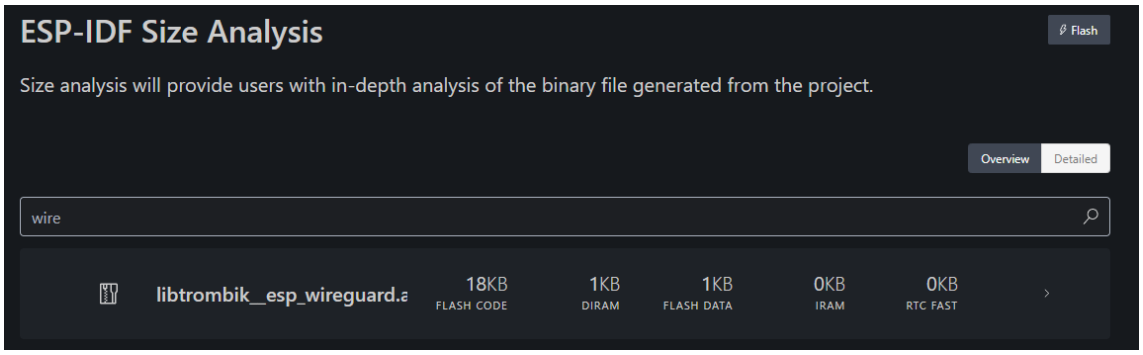
4.1 Analýza paměti

Pomocí Espressif Iot Development Framework rozšíření pro Visual Studio Code můžeme analyzovat podrobně využití paměti celé aplikace, tato funkce je důležitá pro vývojáře, kteří se snaží optimalizovat alokaci paměti a je dostupná v bočním panelu *ESP-IDF: Explorer* pod názvem *ESP-IDF Size* [28], na snímku 4.1 lze vidět přehled využité paměti projektu použitého pro testování ESP32 a WireGuard protokolu.



Obr. 4.1: Přehled paměti aplikace

ESP-IDF dovoluje nahlédnout přesněji do analýzy paměti, přepnutím do detailního režimu lze vidět velikost jednotlivých statických knihoven, viz snímek 4.2.



Obr. 4.2: Detailní přehled paměti aplikace

Jednotlivé knihovny pak mohou být prohlédnuty na úroveň strojových kódů, a jejich individuální alokace, viz tabulka 4.1 pro detailnější rozbor WireGuard knihovny, z tabulky vyplývá, že samotná implementace celkem zabírá skoro 20 kilobajtů.

Objekt	Celková velikost (B)	Flash Code (B)	Flash Data (B)	Diram (B)
wireguard.c.obj	5536	5321	151	64
wireguardif.c.obj	4842	4757	85	0
chacha20.c.obj	1871	1871	0	0
esp_wireguard.c.obj	1757	1360	0	397
blake2s.c.obj	1590	1398	192	0
poly1305-donna.c.obj	1497	1497	0	0
x25519.c.obj	1142	1099	43	0
wireguard-platform.c.obj	817	321	0	496
chacha20poly1305.c.obj	808	744	64	0
crypto.c.obj	62	62	0	0
Velikost regionu	19922	18430	535	957

Tab. 4.1: Seznam velikosti souborů WireGuard knihovny

4.2 Monitorování paměti za běhu programu

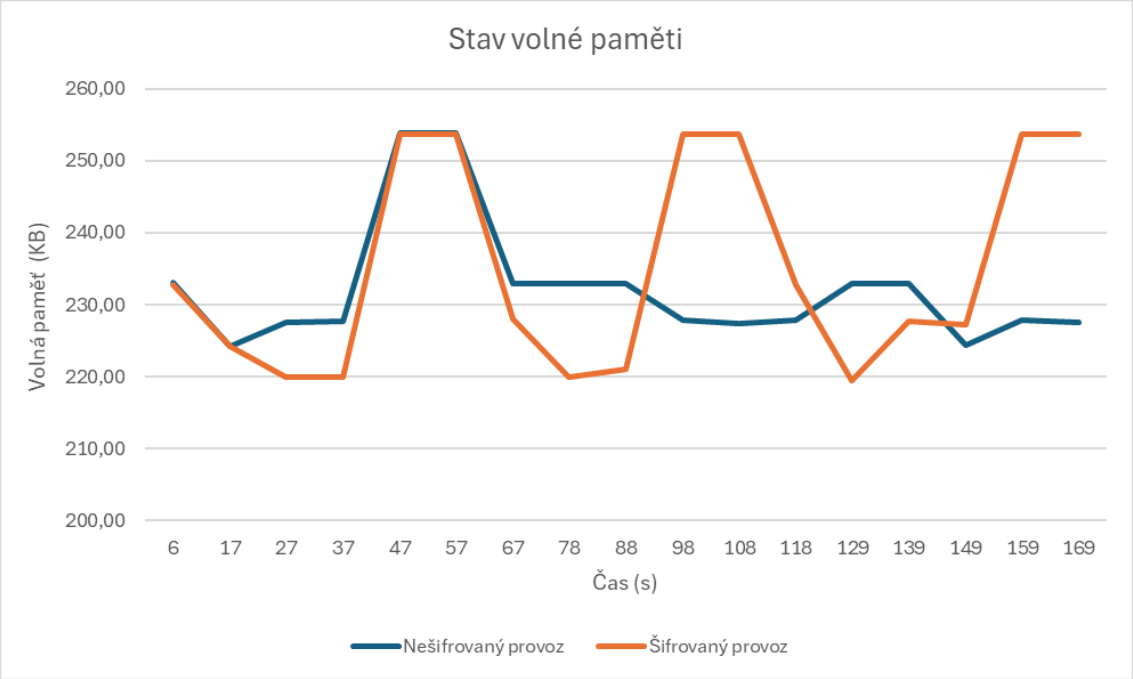
Při provádění Iperf testů, *Task_Monitor* sbíral stav volné paměti *Current heap free size* v intervalu 10 sekund, tyto hodnoty byly zaznamenány do tabulek 4.2, a 4.3, a následně promítnuty do grafu 4.3 k porovnání stavu paměti při šifrovaném a nešifrovaném provozu.

Čas (s)	Volná paměť (KB)	Čas (s)	Volná paměť (KB)
6	232,82	98	253,6
17	224,28	108	253,63
27	220	118	232,81
37	220	129	219,52
47	253,63	139	227,76
57	253,63	149	227,23
67	227,98	159	253,63
78	220	169	253,63
88	221		

Tab. 4.2: Volná paměť při šifrovaném provozu

Čas (s)	Volná paměť (KB)	Čas (s)	Volná paměť (KB)
6	233,09	98	227,89
17	224,27	108	227,34
27	227,54	118	232,92
37	227,75	129	232,99
47	253,79	139	232,99
57	253,79	149	224,44
67	232,98	159	227,93
78	232,95	169	227,55
88	232,95		

Tab. 4.3: Volná paměť při nešifrovaném provozu



Obr. 4.3: Srovnání stavu volné paměti

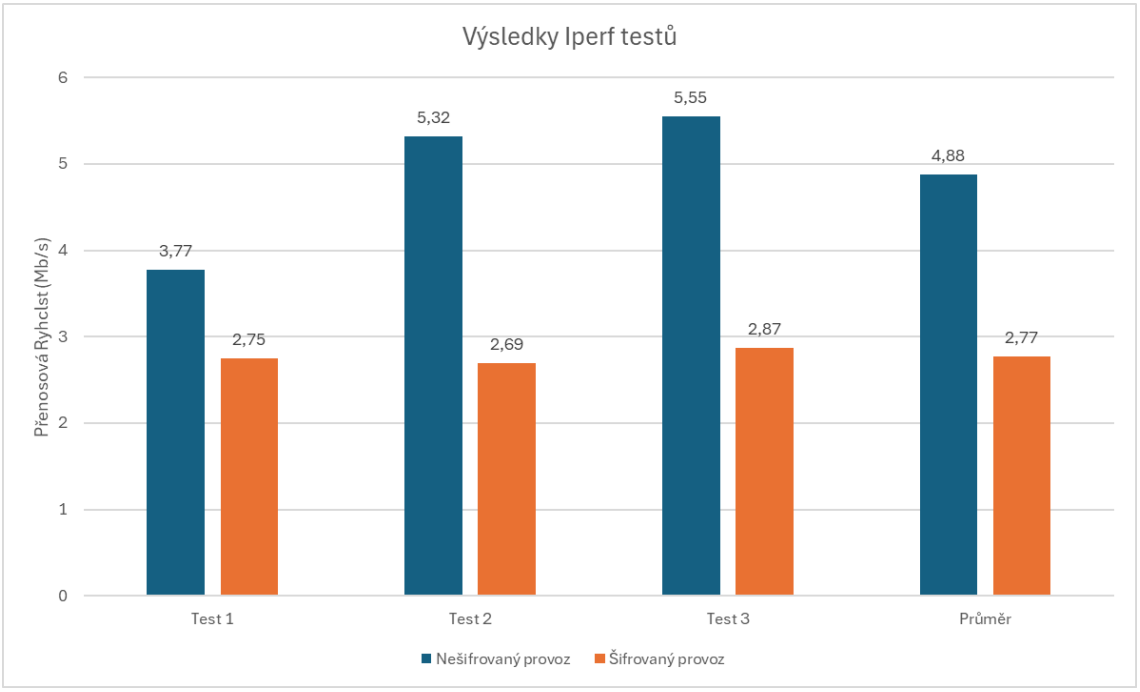
4.3 Testování přenosové rychlosti

Testy přenosové rychlosti byly iniciovány komunikací osobního počítače v roli klienta a ESP32 mikrokontroléru v roli serveru v blízké vzdálenosti mezi sebou. Celkem proběhly 3 testy pro WireGuard a WiFi rozhraní představující šifrovanou a nešifrovanou komunikaci. Jednotlivý test trval 30 sekund s intervalem hlášení každé 3 sekundy, a obě strany měly nastavený buffer socketu na 128 kilobajtů. Výsledná tabulka 4.4 zobrazuje průměrnou přenosovou rychlost v každém testu a celkově, a graf 4.4 názorně zobrazuje rozdíly rychlosti mezi nešifrovaným a šifrovaným provozem.

Pokles rychlosti v šifrovaném provozu lze očekávat, je ale nutno poznamenat, že rychlost šifrovaného provozu je stále obstojná.

Typ provozu	Test 1 (Mb/s)	Test 2 (Mb/s)	Test 3 (Mb/s)	Průměr (Mb/s)
Nešifrovaný	3,77	5,32	5,55	4,88
Šifrovaný	2,75	2,69	2,87	2,77

Tab. 4.4: Výsledky průměrů přenosové rychlosti Iperf testů



Obr. 4.4: Porovnání přenosových rychlostí

Závěr

V práci byla probrána platforma ESP32, vývojová prostředí, nástroje pro testování a jejich implementace na zmíněnou platformu, dále byl probrán WireGuard protokol s obecným přehledem pro vestavěné systémy a následně zaměřením na ESP32 implementaci. Analyzovali jsme alokovanou paměť samotného protokolu. Otestovali jsme výpočetní výkon mikročipu, přenosovou rychlost šifrovaného a nešifrovaného provozu a porovnali jejich rozdíly.

WireGuard se prokázal jako vhodná volba pro zabezpečení vestavěných systémů díky nízkým nárokům na paměť bez ztráty na robustnosti.

Literatura

[1] NEZBEDA, Lukáš. *Implementace a testování protokolu Wireguard.* Online, semestrální práce. Ondřej KRAJSA (vedoucí práce). Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2024. Dostupné z: `https://www.vut.cz/studenti/zav-prace/detail/163766`. [cit. 2025-06-01].

[2] *ESP32.* Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2024. Dostupné z: `https://en.wikipedia.org/wiki/ESP32`. [cit. 2025-06-01].

[3] ESPRESSIF. *ESP32 Series Datasheet.* Online. ESPRESSIF. Espressif.com. 2016, 2025. Dostupné z: `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`. [cit. 2025-06-01].

[4] HORN, Rolf. *Průvodce řadou mikrokontrolerů ESP32.* Online. HW SERVER S.R.O. Vyvoj.hw.cz 2024. Dostupné z: `https://vyvoj.hw.cz/pruvodce-radou-mikrokontroleru-esp32.html`. [cit. 2025-06-01].

[5] NABTO. *ESP32 for IoT: A Complete Guide.* Online. NABTO. Nabto.com Dostupné z: `https://www.nabto.com/guide-to-iot-esp-32/`. [cit. 2025-06-01].

[6] ITNETWORK. *Seznámení s ESP-32.* Online. ITNETWORK. Itnetwork.cz. Dostupné z: `https://www.itnetwork.cz/hardware-pc/arduino/esp32/seznameni-s-esp-32`. [cit. 2025-06-01].

[7] ESPRESSIF. *Modules.* Online. ESPRESSIF. Espressif.com. Dostupné z: `https://www.espressif.com/en/products/modules`. [cit. 2025-06-01].

[8] POŠTULKA, Jiří. *Programování mikrokontrolérů ESP32.* Diplomová práce, vedoucí Marek Babiuch. Ostrava: Vysoká Škola Báňská – Technická univerzita Ostrava, Fakulta strojní, Katedra automatizační techniky a řízení, 2020. Dostupné také z: `https://core.ac.uk/download/328787655.pdf`. [cit. 2025-06-01].

[9] ARDUINO. *Arduino-ide.* Online. GITHUB. GitHub.com. 9 April 2025. Dostupné z: `https://github.com/arduino/arduino-ide`. [cit. 2025-06-01].

[10] ESPRESSIF. *Frequently Asked Questions.* Online. ESPRESSIF. Espressif.com. Dostupné z: `https://docs.espressif.com/projects/arduino-esp32/en/latest/faq.html`. [cit. 2025-06-01].

[11] DRONEBOT WORKSHOP. *Getting Started with PlatformIO.* Online. DRONEBOT WORKSHOP. Dronebotworkshop.com. Dostupné z: `https://dronebotworkshop.com/platformio/`. [cit. 2025-06-03].

[12] POSITROID BLOG. *Enabling runtime statistics collection in ESP32-arduino.* Online. POSITROID BLOG. Positroid.tech. 2025. Dostupné z: https://positroid.tech/en/post/platformio-esp32-stats. [cit. 2025-06-01].

[13] ESPRESSIF. *Project Configuration.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/kconfig.html. [cit. 2025-06-01].

[14] ESPRESSIF. *IDF Frontend.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/tools/idf-py.html. [cit. 2025-06-01].

[15] ESPRESSIF. *Build System.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/build-system.html. [cit. 2025-06-01].

[16] SOURCEFORGE. *IPERF.* Online. SOURCEFORGE. sourceforge.io. March 2025. Dostupné z: https://iperf2.sourceforge.io/iperf-manpage.html. [cit. 2025-06-01].

[17] DONENFELD, Jason A. *WireGuard: Next Generation Kernel Network Tunnel.*. Online. Wireguard.com. Dostupné z: https://www.wireguard.com/papers/wireguard.pdf. [cit. 2025-06-02].

[18] HOPE, Daniel. *Wireguard-lwip.* Online. GitHub. Github.com. 18 January 2021, 29 May 2022. Dostupné z: https://github.com/smartalock/wireguard-lwip. [cit. 2025-06-02].

[19] SWEETING, Nick. *Wireguard-docs.* Online. GitHub. Github.com. 28 May 2019, 24 November 2024. Dostupné z: https://github.com/pirate/wireguard-docs. [cit. 2025-06-02].

[20] INTROSERV. *WireGuard Windows setup.* Online. INTROSERV. Introserv.com. Dostupné z: https://introserv.com/docs/wireguard-windows-setup/. [cit. 2025-06-02].

[21] HOPE, Daniel; IDA, Kenta; DEMPSKY, Matthew a D. J., Bernstein. *Esp_wireguard.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://components.espressif.com/components/trombik/esp_wireguard/versions/0.9.0. [cit. 2025-06-02].

[22] ESPRESSIF. *Iperf.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://components.espressif.com/components/espressif/iperf/versions/0.1.3. [cit. 2025-06-02].

[23] ESPRESSIF. *Iperf-cmd.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://components.espressif.com/components/espressif/iperf-cmd/versions/0.1.3. [cit. 2025-06-02].

[24] PAVLUSHA, Volodymyr. *ESP32_Task_Monitor.* Online. GITHUB. Github.com. Dostupné z: https://github.com/VPavlusha/ESP32_Task_Monitor. [cit. 2025-06-02].

[25] ESPRESSIF. *Heap Memory Allocation.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/system/mem_alloc.html. [cit. 2025-06-02].

[26] EEMBC. *Coremark.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://components.espressif.com/components/espressif/coremark/versions/1.1.0~1. [cit. 2025-06-03].

[27] EEMBC. *CoreMark.* Online. EEMBC. Eembc.org. Dostupné z: https://www.eembc.org/coremark/. [cit. 2025-06-03].

[28] ESPRESSIF. *ESP-IDF Application Size Analysis.* Online. ESPRESSIF. Espressif.com. Dostupné z: https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/additionalfeatures/application-size-analysis.html. [cit. 2025-06-03].

Seznam symbolů a zkratek

Seznam příloh