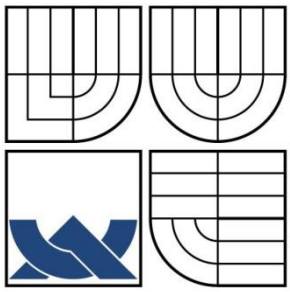
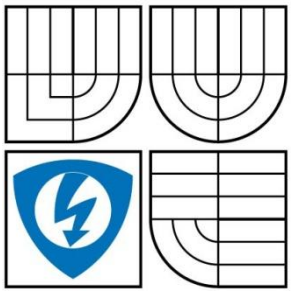


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ZABEZPEČENÝ PŘEVODNÍK STANDARDU RS232 NA INTERNET

SECURE CONVERTER FOR AN RS-232 STANDARD TO THE INTERNET

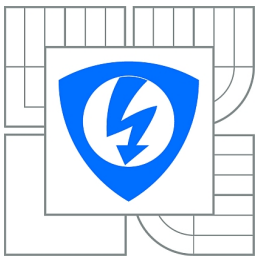
DIPLOMOVÁ PRÁCE

MASTER´S THESIS

AUTOR PRÁCE *BC. MICHAL POKORNÝ*
AUTHOR

VEDOUCÍ PRÁCE *ING. MARTIN KOUTNÝ*
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Michal Pokorný ***ID:*** 77730
Ročník: 2 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Zabezpečený převodník standardu RS-232 na Internet

POKYNY PRO VYPRACOVÁNÍ:

Podrobně se seznamte s aktuálními bezpečnostními standardy a jejich podporou v síti Internet. Na základě výběru pak navrhněte a realizujte zabezpečený převodník standardu RS-232 na Internet s moduly RCM2700. K tomuto účelu vytvořte aplikační podporu pod OS Windows XP.

DOPORUČENÁ LITERATURA:

[1] MACKAY, Steve, et al. Practical Industrial Data Networks : Design, Installation and Troubleshooting. [s.l.] : Newnes, 2003. 576. ISBN 978-0750658072.
[2] PIRKL, Josef. Síťové programování pod Windows a programování Internetu. 2002. 360s. ISBN 80-7232-145-5.
[3] Rabbit Semiconductor : RCM3700 RabbitCore [online]. 2008 [cit. 2008-04-24]. DostupnA1 z WWW: <http://rabbit.com/products/rcm3700/docs.shtml>.

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 26.5.2010

Vedoucí práce: Ing. Martin Koutný

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Anotace

Diplomová práce řeší problém vytvoření bezpečného přenosového kanálu mezi virtuálními sériovými porty na osobním počítači a sériovými porty na vývojovém KITu Rabbit RCM3700.

Dnešní komunikační cesty neposkytují náležité zabezpečení celé komunikace. Proto je na účastnících, aby toto zabezpečení realizovali sami. Tímto zabezpečením se rozumí zajištění spolehlivého přenosu dat, která budou mezi účastníky zašifrována tak, aby případný útočník je nebyl schopen v reálném čase číst, popřípadě měnit.

Výsledkem diplomové práce je návrh a realizace, která kromě šifrování dostatečně bezpečným algoritmem zajišťuje i ověření autenticity mezi účastníky komunikace. Jako šifrovací algoritmus byl zvolen hojně používaný algoritmus AES, a jako autentizační algoritmus zvolen algoritmus, který pro ověření autenticity vyžaduje znalost tajného klíče.

Klíčová slova: virtuální sériový port, Rabbit, RCM3700, Ethernet, AES, kryptografie, autentizace

Abstract

Master´s thesis tries to find a solution to make a secure transmission channel between virtual serial ports on the personal computer and the serial ports on the Rabbit RCM3700 development KIT.

Today’s communications channels don't offer appropriate security of a whole communication. Therefore it depends on get-togethers, in order to realize this security themselves. This security means ensuring reliable transmission of data to be encrypted between parties so that any attacker is not able to read real-time, or eventually change them.

As a result of this Master thesis is the design and implementation, which in addition to encryption algorithm provides sufficient security and authenticity of communication between the parties. As an encryption algorithm has been chosen widely used AES algorithm and as authentication algorithm has been chosen algorithm, which for authenticity requires knowledge of the secret key.

Keywords: virtual serial port, Rabbit, RCM3700, Ethernet, AES, cryptography, authentication

POKORNÝ, M. *Zabezpečený převodník standardu RS-232 na Internet .* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 70 s. Vedoucí diplomové práce Ing. Martin Koutný.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Zabezpečený převodník standardu RS-232 na Internet“ jsem 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 diplomové 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 § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Martinu Koutnému, za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce. Dále bych chtěl poděkovat svým rodičům za psychickou a finanční podporu poskytovanou po dobu mého studia.

V Brně dne ....

podpis autora

Obsah

Úvod

Výstupem diplomové práce je realizace komunikačních kanálů, kterými lze přenášet data mezi jednotlivými sériovými porty prostředí Rabbit RCM3700 a virtuálními sériovými porty v osobním počítači. Při návrhu i realizaci bylo nutno přihlédnout k faktu, že přenášená data budou přenášena nechráněnými sítěmi, jako je například Internet. Z těchto důvodů bylo třeba komunikaci odpovídajícím způsobem zabezpečit.

V rámci diplomové práce byl nejprve vytvořen návrh celého konceptu. V tomto návrhu bylo nejprve rozhodnuto, jakým způsobem budou data mezi sériovými porty přenášena, a jakých protokolů a algoritmů bude s největší pravděpodobností použito.

Podle již zmíněného návrhu byla provedena realizace aplikace pro operační systém Microsoft Windows. Aplikace byla vytvořena takovým způsobem, aby byla schopná vytvořit současně více virtuálních sériových portů a vytvořit mezi nimi bezpečné přenosové kanály. Práce s takto vytvořeným virtuálním sériovým portem byla realizována asynchronně tak, aby nedocházelo k přílišnému a částečně i zbytečnému zatížení procesoru. Data mezi jednotlivými účastníky komunikace jsou šifrována dostatečně bezpečným algoritmem tak, aby případný útočník nebyl schopen tato data číst, měnit nebo nějakým jiným způsobem zneužít. Celý model síťové komunikace je založen na konceptu klient – server, u kterého se na začátku komunikace ověří identita komunikující strany a to jak ze strany klienta, tak i serveru.

Po dostatečném odladění aplikace bylo přikročeno k realizaci programu pro vývojové prostředí Rabbit. Díky omezeným možnostem řídícího 8bitového mikroprocesoru bylo obtížnější realizovat některé algoritmy nebo komunikační protokoly. Zároveň s laděním tohoto programu byly řešený dříve neobjevené problémy v programu pro osobní počítač. Po konzultaci s vedoucím práce byla přidána možnost nastavovat parametry komunikace na systému Rabbit pomocí webového rozhranní.

Při návrhu i realizaci byl brán ohled na to, že osobní počítač a vývojové prostředí Rabbit RCM3700 jsou poměrně odlišných počítačových architektur. Osobní počítač je založen na Von Neumannově architektuře, zatímco vývojové prostředí má prvky Harvardské

architektury. Z těchto důvodů musel být brán ohled na vlastnosti a schopnosti obou systémů.

Rozdílnost systémů, na kterých se prováděla implementace komunikačních kanálů, do značné míry ovlivnila i výběr vhodných protokolů a algoritmů. Touto rozdílností byl například ovlivněn i výběr protokolu na síťové vrstvě nebo výběr odpovídajících vhodných algoritmů.

Teoretický úvod

Cílem této kapitoly je seznámení se s poznatky a termíny, které je třeba znát před realizací přenosového kanálu realizujícího přenos mezi virtuálními a reálnými sériovými porty standartu RS-232.

1.1 Úvod do zpracování informace

Současná společnost je závislá na informačních technologiích, které umožňují rychlé a přesné zpracování informací v rozmanitých oblastech lidské činnosti. Některé oblasti by dokonce na své současné úrovni ani nemohly existovat. Mezi tyto odvětví můžeme zařadit například kryptografii, hromadné zpracování dat nebo komprese digitalizovaných dat.

Na počátku vzniku dnešních počítačů bylo třeba vytvořit a zavést jednotný způsob pro práci s informací. Z toho důvodu vznikla jednotka jeden bit. Tato jednotka může obsahovat pouze dva přesně definované stavy a to logickou nulu a logickou jedničku. Pro práci s bloky dat byla zavedena jednotka byte (čte se bajt), která se skládá z osmi bitů. S rozmachem technologií bylo třeba ukládat daleko větší množství dat, a proto vznikly jednotky 1 kB (1024 Bytů), 1 MB (1024 kB) a samozřejmě i větší jednotky.

Na takto definovaných a uznávaných jednotkách jsou založeny všechny dnešní digitální obvody a tím pádem i dnešní digitální počítače. Tato vlastnost nám umožňuje jednotné zacházení s digitálně uloženou informací, díky čemuž můžeme propojovat různorodá zařízení a přenášet mezi nimi digitální informace neboli data. Podrobněji se úvodem do výpočetní techniky zabývá zdroj [1].

1.2 Ukládání hodnot do paměti – malý a velký indián

Při ukládání hodnot je důležitým aspektem způsob, jak budou hodnoty v paměti rozloženy. Pokud je třeba uložit hodnotu, která se vleze do jednoho bytu, je situace jednoduchá. Problém nastává, pokud se hodnota ukládá do více bytů. Bohužel v této oblasti nedošlo k jednotné standardizaci a místo jednoho způsobu uložení máme dva[2].

Prvním způsobem je tzv. malý indián (z anglického little endian). V tomto případě jsou nižší byty ukládány na nižší adresu v paměti. Tento způsob používají například nejrozšířenější počítače založené na architektuře Intel x86.

Druhým způsobem je tzv. velký indián (z anglického big endian). Takto uložené hodnoty jsou uloženy zrcadlově k prvnímu způsobu, a to tak, že vyšší byty jsou ukládány na nižší adresu v paměti. Tento způsob je užíván například firmou Freescale (dříve Motorola) nebo se s ním dá setkat při práci se síťovým rozhraním počítačem.

Z těchto důvodů je třeba si při práci na různorodých architekturách a systémech dávat pozor na to, aby nedocházelo k mylným interpretacím uložených hodnot.

1.3 Komunikační metody

Při komunikaci mezi jednotlivými koncovými body komunikace je třeba zajistit, aby při komunikaci nedocházelo k zbytečným ztrátám informací a aby příjemce komunikace obdržel informace včas a nepoškozené. Z těchto důvodů je potřeba, aby se všechny body podílející se na komunikaci s ostatními body komunikace chovaly podle předem definovaných pravidel či standardů.

Dosti důležitým parametrem při hodnocení komunikace mezi jejími body je přístup na společné přenosové médium. Existují dvě hlavní skupiny metod. První je deterministická, která vylučuje kolize při přístupu na společné přenosové médium. Tyto metody jsou propracované a vyžadují od všech bodů komunikace dodržovat přesně stanovená pravidla komunikace. Na rozdíl od deterministických metod existují metody stochastické, které nevylučují, že může kolize nastat. Pro případ, že kolize nastane, musí mít všechny body komunikace implementováno, jak na tento stav reagovat a jak obnovit běžnou komunikaci na přenosovém médiu. V případě, že komunikace neprobíhá po jednom sdíleném médiu, ale pro každé propojení jednotlivých bodů komunikace včetně jednotlivých směrů existuje nesdílená přenosová cesta, nemusí se koliznost a nekoliznost na přenosovém médiu vůbec řešit.

Dalším velmi důležitým parametrem komunikace je synchronnost či asynchronnost přenosu dat. Synchronní přenos se vyznačuje tím, že s přenosem informace po vodičích je nutno v dalším vodiči přenášet synchronizaci přenášených informací. Tímto

způsobem je zaručeno, že nedojde k špatné interpretaci přenášených informací. Oproti tomu asynchronní přenos nevyžaduje další vodič nebo vodiče pro synchronizaci, ale synchronizaci provádí se začátkem přenosu oba konce synchronizace tj. příjemce i odesilatel. V okamžiku, kdy vysílající začne vysílat informace na přenosové médium, tak spustí ve vysílací části synchronizační takt, který řídí vkládání informace na přenosové médium. Rovněž pak na přijímací části v okamžiku, kdy je detekován příjem dat, je spuštěn synchronizační takt, který řídí okamžik čtení dat z přenosového média.

Přenosová rychlost je kromě taktovací frekvence vysílače a přijímače výrazně závislá na množství současně přenášených informací. Množství lze navýšit třeba vícestavovou modulací nebo navýšením přenosových cest. Pro počet současně přenášených informací existuje rozdělení na sériové, paralelní a sérioparalelní.

První jmenované umožňují v jednom okamžiku přenášet pouze jednu informaci. Navýšit přenosovou informaci proto jde pouze zvýšením taktovací frekvence nebo navýšením počtu současně přenášených stavů na jednom vodiči.

Na rozdíl od toho paralelní přenos umožňuje přenos v jednom okamžiku tolika vodiči, jaká je šířka zpracovávaného slova koncového bodu. Díky navýšení počtu vodičů se přenosová rychlost navýší o násobek, o který se navýšil počet vodičů. Tento způsob navýšení má dvě výrazné nevýhody a to navýšení ceny přenosového vedení a nutnost zajistit, aby nedocházelo k nežádoucím interferencím mezi sousedícími přenosovými cestami.

Třetím způsobem přenosu je sérioparalelní přenos. Při něm se nepřenáší celá šířka koncovým bodem zpracovávaného slova, ale pouze jeho část. Vysílač pak musí být schopen rozdělit šířku slova a vyslat ji po částech a přijímač pak musí být schopen takto rozdělené slovo zpracovávaných informací opět složit.

1.4 Typy komunikačních médií a metod

S rozvojem současné techniky vzniklo mnoho komunikačních médií a standardů, které definují, jak by se měla tato média využívat, jaké parametry by se měly při využívání

dodržovat a rovněž i zásady, aby nedošlo k poškození samotného média nebo majetku či zdraví osob a zvířat.

V současnosti tedy existuje více typů médií, jež se od sebe liší svými fyzikálními vlastnostmi, nosiči informace v tomto médiu a prostředím, ve kterém se tyto nosiče informace pohybují.

Prvním a také nejpoužívanějším médiem je metalické vedení. Nosičem informace v tomto případě jsou elektrony, jejichž směr proudění neboli elektrický proud je využit v definování stavů, které se pak převádí na logické informace, s kterými umí pracovat výpočetní logika v ukončovacích bodech přenosových cest. Metalická vedení se dají rozdělit na jednocestné a vícecestné. U jednocestné je v jednom fyzickém vedení pouze jeden vodič, kterým se přenáší informace. Naproti tomu u vícecestné je v jednom vedení více vodičů, které přenášejí informace mezi jednotlivými body komunikace. Díky této skutečnosti jsou schopny za stejnou jednotku času přenést více informací než jednocestná média. U metalických vedení je třeba kvůli vznikajícímu elektromagnetickému poli zajistit, aby byly jednotlivé vodiče od sebe dostatečně odděleny tak, aby přenos informace v jednom vodiči neovlivňoval přenos v druhém. Toto elektromagnetické oddělení výrazně ovlivňuje cenu vícecestného vedení a proto se od nich v místech, kde je potřeba ekonomicky náročný a zároveň velmi spolehlivý přenos, upouští.

Dalším důležitým médiem pro přenos informace je přenos informace pomocí elektromagnetického pole. Tento způsob přenosu se využívá na místech, kde není možno vést fyzické vedení. Nevýhodou tohoto způsobu je jeho snadné rušení nebo odposlouchávání přenášených informací. Přímo pro tento způsob přenosu vzniklo mnoho šifrovacích metod, které mají zabránit nežádoucím osobám přístupu k přenášeným informacím.

Posledním významným médiem pro přenos informace je přenos fotonu mezi účastníky komunikace. Tomuto způsobu přenosu se říká optický. Na straně vysílače je potřeba převést elektrický signál na světelný tok. Nejčastěji se zde používá laserová dioda, která je díky svým vlastnostem pro tento účel ideální. Na straně přijímače je opět potřeba převést světelný tok na elektrický signál. K tomu se nejčastěji používají

fotocitlivé polovodičové prvky. Jako médium se nejčastěji užívá volný prostor nebo je světelný paprsek přenášen po optických kabelech. V prvním případě není třeba budovat nějakou infrastrukturu. Jedinou podmínkou je přímá viditelnost mezi vysílačem a přijímačem. Nevýhodou použití volného prostoru může být velká závislost na počasí, kdy při mlze nebo hustém dešti může docházet k velkému útlumu světelného paprsku a tudíž i zhoršení nebo znemožnění komunikace. V případě použití optického kabelu jsou sice vyšší náklady na vybudování infrastruktury, ale díky vedení světla ve světelně neměnném prostředí není ohrožena stabilita spoje. Mezi nevýhody patří vyšší cena vybudování infrastruktury a nutnost dodržení celkem striktních požadavků na vedení optického kabelu, jehož jádro je buď ze speciálního skla nebo ohebného plastu. Oproti předešlým dvou přenosovým médiím jsou optické spoje odolné vůči elektromagnetickému rušení. Optickými kabely se zabývá zdroj [3].

1.5 RS-232C

Tento standard vznikl již v roce 1969. I přes to je díky své jednoduchosti hojně používán, i když ho v posledních letech začaly nahrazovat modernější a mnohem rychlejší standardy jako je USB či FireWire. Jeho implementaci lze najít jak v osobním počítači (v současnosti ho však už víceméně nahradil standard USB), tak například v mikrokontrolerech, síťových prvcích nebo i v běžné domácí elektronice, kde například slouží k aktualizaci vnitřního software zařízení.

Plná implementace tohoto standardu obsahuje dva samostatné kanály, které umožňují příjem i vysílání sériových dat. Častěji se však lze setkat s částečnou implementací, tj. pouze s jedním kanálem. Zapojení tohoto zjednodušeného konektoru je vidět v následující tabulce. Z pojmenování je vidět původní účel tohoto standardu a to pro připojení modemu k počítači.

Jak je vidět z tabulky číslo jedna pin TxD slouží pro vysílání dat, kdežto pin RxD je určen pro příjem dat. Dalším důležitým pinem je GND. Ostatní piny nejsou pro přenos dat nezbytně důležité. Data jsou vysílána pomocí dvou úrovní a to logické úrovně L a H. Úroveň L je nízká úroveň a odpovídá ji 12 voltů. Logické úrovni H je vysoká úroveň a odpovídá ji 12 voltů na vodiči.

Tabulka 1: Popis vývodů jednoho kanálu RS-232

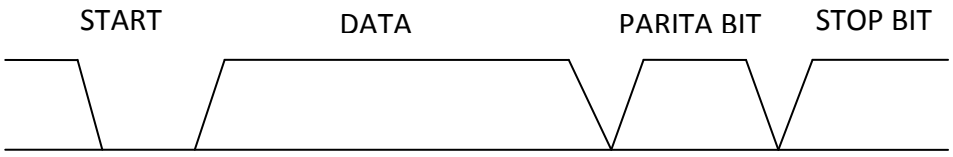
Vývod	Název	Vstup/výstup	Význam
1	DCD	vstup	Detektor nosného signálu
2	RxD	vstup	Přijímaná data
3	TxD	výstup	Vysílaná data
4	DTR	výstup	Pohotovost koncového zařízení
5	GND	-	Signálová zem
6	DSR	vstup	Pohotovost ukončujícího zařízení
7	RTS	výstup	Výzva k vysílání
8	CTS	vstup	Pohotovost k vysílání
9	RI	vstup	Indikátor volání

Před začátkem přenosu musí být nejprve nastaveno několik parametrů. Nejdůležitějším je přenosová rychlost, počet přenášených bitů, parita a počet stop-bitů.

Zařízení podle tohoto standardu umožňují asynchronní přenos. Tato zařízení se pak označují zkratkou UART. Pokud je součásti implementace i synchronní přenos informace, je označován USART .

Výhodou tohoto standardu jsou vysoká odolnost vůči zkratu a dovolují odebírat proud z portů až 10mA. Díky tomu lze přímo za chodu počítače nebo jiného zařízení k těmto portům připojovat či odpojovat další počítače nebo jiné zařízení.

Na následujícím obrázku, který byl přejat ze zdroje [7] je vidět průběh rámce. Tento rámec začíná start bitem a končí stop bitem. Hned po odvysílání start bitu jsou odeslána data a za nimi následuje paritní bit a stop bit.



Obrázek 1: Tvar rámce standardu RS232

Dalšími informacemi ohledně standardu RS-232 se zabývají zdroje [1], [4], [5], [6] a rovněž i katalogové listy k různým převodníkům napětí atd., které lze získat na adrese [7].

1.6 Síťové rozhraní

I když by se na první pohled mohlo zdát, že síťová rozhraní jsou trendem posledních dvou desetiletí, je jejich vznik datován do šedesátých let dvacátého století. V této době docházelo k propojování velkých sálových počítačů, které v té době patřily především univerzitám a armádním složkám. První takovou počítačovou sítí byla síť ARPANET, která propojovala počítače na území Spojených států amerických.

Za poslední desetiletí tato síťová rozhraní prošla velmi dramatickým vývojem a vzniklo mnoho standardizovaných typů. Mnoho z nich zůstalo pouze u experimentálních prototypů nebo se je nepodařilo rozšířit natolik, aby se mohlo o jejich nasazení vážněji uvažovat. V mnoha případech o jejich neúspěšnosti nerozhodla technologická nedokonalost, ale třeba ekonomické faktory nebo licenční politika jejich autorů.

V současnosti existuje mnoho různorodých standardů i funkčních síťových zařízení. Mezi ty nejvýznamnější patří Ethernet, Token Ring, FDDI, Wi-Fi nebo ATM. Pro vytvoření domácí počítačové sítě nebo lokální sítě v malé nebo střední firmě jsou nejpoužívanější Ethernet a Wi-Fi jako jeho bezdrátová náhrada.

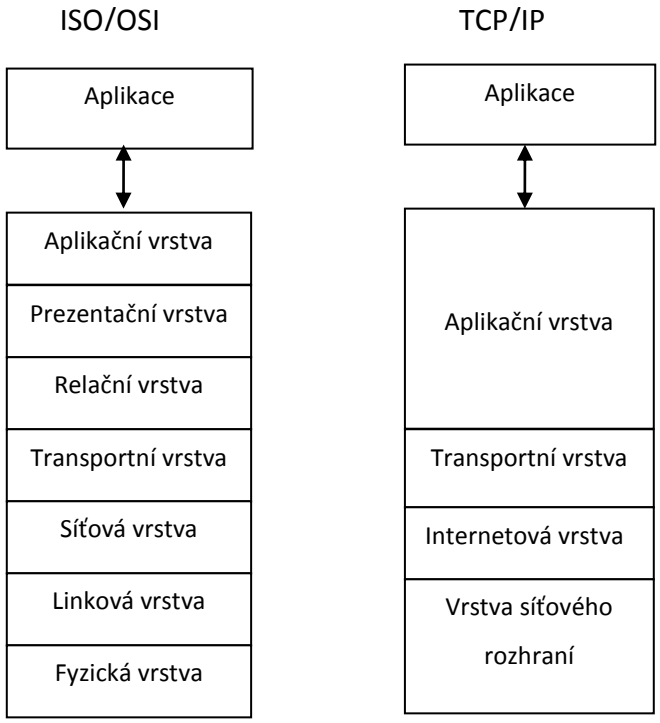
1.6.1 Vrstvy síťových modelů

V době vzniku prvních počítačových sítí vznikaly i aplikace, které umožňovaly uživateli práci v rámci sítě. Díky absenci standardů v době vzniku prvních sítí existovalo mnoho různorodých aplikací. Bohužel každá aplikace byla řešena bez nějakého jednotného vzoru, a tudíž kompatibilita mezi aplikacemi na různých sítích neexistovala. Z těchto důvodů bylo vytvořeno několik modelů, které by vedly k větší jednotnosti a tím pádem vyšší kompatibilitě aplikací, které v rámci těchto sítí fungovaly. Tím nejdůležitějším, o které se opíraly všechny pozdější je ISO/OSI.

Tento model obsahuje 7 vrstev. Každá z vrstev kromě fyzické by měla vědět, jaké služby může požadovat od vrstvy bezprostředně nižší a každá z vrstev kromě aplikační by měla vědět, jaké služby musí poskytovat služby bezprostředně vyšší. Rozložení jednotlivých vrstev je vidět na následujícím obrázku číslo 1.

Účelem aplikační vrstvy je poskytnout služby modelu běžící aplikaci. Díky tomu aplikaci stačí znát pouze několik málo funkcí a díky nim může vytvářet a rušit spojení, vysílat

a přijímat data a provádět další operace, které ji jsou pomocí aplikační vrstvy k dispozici.



Obrázek 2: Porovnání referenční modelu ISO/OSI a modelu TCP/IP

Prezentační vrstva by se měla starat o převod dat do patřičného kódování, pokud je to potřeba, tak by měla provádět kompresy či dekompresy dat a pokud je vyžadováno šifrování tak i šifrování a dešifrování dat.

O udržování relace mezi dvěma a více koncovými body se stará relační vrstva. Jejím úkolem je zajistit roztřídění přenášených dat tak, aby data přijatá ze společné přenosové cesty byla následně podle předem daných pravidel rozřazena podle toho, k jaké relaci přísluší.

V modelu ISO/OSI je úkolem transportní vrstvy zajistit určitou úroveň kvality přenosu dat. Víceméně je jejím úkolem vyrovnávat různou kvalitu a vlastnosti přenosových sítí. Dalším dosti důležitým úkolem je adresace v rámci jednoho počítače. Toto se děje pomocí takzvaných portů. Model ISO/OSI byl navržen s tím, že na každé ze 7 vrstev modulu může být použit jakýkoliv protokol poskytující své služby vrstvě

nad a využívající služeb nižší vrstvy. Díky tomuto může být na transportní vrstvě použito více protokolů. V rámci pak konkrétního protokolu existuje množina použitelných portů. Z těchto důvodů je poté potřeba při adresaci určit, jakému transportnímu protokolu je zpráva určena.

Pro adresování v rámci sítě je velmi důležitá vrstva síťová. Protokoly pracující na této vrstvě se starají výhradně o adresaci zpráv mezi uzly sítě. Dosti důležité jsou implementace na uzlech sítě, které se starají o propojení jednotlivých koncových uzlů nebo celých sítí. Na těchto uzlech musí být určitým způsobem definováno, jak naložit s přenášenými daty. Díky činnosti, kterou provádějí, se dosti často tyto propojovací prvky označují jako směrovače.

Linková vrstva je na rozdíl od předešlých vrstev daleko více závislá na použitém zařízení. Mezi hlavní úkoly této vrstvy patří řízení přístupu k přenosovému médiu, adresace v rámci hardwarových adres konkrétního zařízení a přímo provádí příjem nebo odesílání dat na přenosové médium. Zařízením, které slouží pro propojení koncových uzlů na této vrstvě, se podle jejich funkce říká buď směrovače, nebo rozbočovače. Pokud má rozbočovač pouze dva porty, říká se mu opakovač.

Účelem fyzické vrstvy je samotný přenos dat. Stará se o to, jak by měla vypadat struktura přenášených dat, jak by měly být reprezentovány logické úrovně na přenosovém médiu. Dále pak definuje mechanickými vlastnostmi jako je typ, vlastnosti kabeláže a konektorů pro propojení.

Referenční model ISO/OSI je považován jako modelový díky svým výhodám, jako je přehlednost, modulárnost při potřebě vyměnit jeden protokol za jiný, menší složitost sítí a zvýšená míra specializace tvůrců protokolů či zařízení pracující na jedné konkrétní vrstvě. Tyto výhody se v reálném prostředí dnešních sítí a výpočetní techniky nevyužije v plné míře. Proto vzniklo několik modelů, které některé tyto vrstvy buď neimplementují vůbec, nebo jsou implementovány v rámci vrstvy, která obsahuje implementace i některé další vrstvy nebo vrstev. V dnešní době prakticky nejpoužívanějším modelem je vrstvový model TCP/IP. Tento model lze vidět na obrázku číslo 2, který je výše. Tento model má 4 vrstvy, které vycházejí z referenčního modelu.

Aplikační vrstva ve vrstvovém modelu TCP/IP odpovídá vrstvám aplikační, prezenční a relační v referenčním modelu ISO/OSI. Díky tomuto spojení došlo k úspoře procesorového času při předávání dat mezi těmito třemi vrstvami. Na druhou stranu je na protokolech aplikační vrstvy, aby implementovaly funkce, které by jinak byly implementovány v těchto vrstvách.

V transportní vrstvě pracují v modelu TCP/IP nejčastěji dva protokoly a to TCP a UDP. Prvně jmenovaný je spojově orientovaný se zajištěním spolehlivého doručení přenášených dat. Tento protokol tedy pro vyšší vrstvu zajišťuje spojitý proud dat s tím, že jestliže dojde ke ztrátě nebo poškození, dokáže to tento protokol zjistit a pokusí se o nápravu. Nevýhodou TCP je vyšší režie při přijmu nebo odesílání dat. TCP se většinou používá tam, kde je třeba zajistit spolehlivé doručení dat. Naproti tomu UDP je nespojitá služba, která nijak neošetřuje poškození nebo ztrátu přenášených dat. Tato detekce a případná náprava je přenechána na vyšší vrstvě, která kontrolu či opravení musí, pokud ji vyžaduje, sama implementovat. Její výhodou je pak nižší režie a tím pádem i vyšší rychlost při odesílání nebo přijímání přenášených dat. Jednotkou informace na této vrstvě, stejně jako v referenčním modelu na odpovídající vrstvě, je segment.

Internetová vrstva v modelu TCT/IP odpovídá vrstvě sítové v referenčním modelu ISO/OSI. Jak již bylo uvedeno výše, úkolem této vrstvy je směrovat datové jednotky od zdroje k cíly. Jednotkám na této vrstvě pak říká datagramy. V dnešních počítačích se velmi často na této vrstvě používá protokol IP. Tento protokol má několik verzí. Některé tyto verze se bohužel nedočkaly ostrého nasazení nebo byly nasazeny jen v omezené míře pro testovací účely. Nejrozšířenější a nejznámější verzí je verze 4. Ta je například použita v největší síti světa, Internetu, jako majoritní protokol pro směrování v rámci této sítě. Bohužel v době vzniku tohoto protokolu nikdo nepředpokládal tak vysoký počet připojených uživatelů do této sítě. Následně vznikaly další verze tohoto protokolu, které nedostatek adres řeší. Nástupcem verze 4 má být verze 6, která mimo to přináší mnoho dalších novinek.

Poslední, nejspodnější vrstvou v tomto modelu, je vrstva síťového rozhranní. Tato vrstva odpovídá v modelu ISO/OSI dvou vrstvám a to vrstvě linkové a fyzické. Proto

tato vrstva implementuje přístup k přenosovému médiu, hardwarovou adresaci i technické a elektrické parametry fyzického provedení. V dnešních počítačích s moderním operačním systémem je tato vrstva často realizována jako kombinace síťové karty a příslušného ovladače zařízení.

1.6.2 IP

Jak již bylo uvedeno v předešlé podkapitole, IP neboli Internet Protocol, je protokol na internetové vrstvě, který řeší adresaci mezi jednotlivými uzly sítě. K tomu, aby mohl jednoznačně identifikovat jeden konkrétní uzel a na něm jedno konkrétní rozhraní, musí definovat určitý adresní prostor a adresy z tohoto prostoru pak přidělit jednotlivým síťovým rozhraním.

V adresním prostoru protokolu IP existuje několik speciálních adres nebo skupin adres, kterým se říká podsítě. Existuje několik typů těchto podsítí a to například lokální adresy, adresy sítě, všesměrové, multicastotové a nebo adresy určené pro demonstrační účely. Díky tomu, že různé verze protokolů mají jinou šířku adresního prostoru, mohou mít tyto skupiny adres jinou hodnotu i přes to, že jsou použity pro stejné účely.

Verze číslo 4 disponuje 32 bitovou adresou a díky tomu lze adresovat až 2^{32} unikátních adres. V této verzi byly použity takzvané třídy adres. Každá z těchto tříd definovala, jakou masku při směrování má použít. Se zaplňujícím se adresním prostorem byla potřeba použít masku, která by byla daleko flexibilní, aby při použití jedné velké třídy nedocházelo k plýtvání, pokud nejsou adresy v tomto přiřazeném adresním prostoru zcela využity. Flexibilní maska bohužel vede k větší segmentaci, a tím pádem i větším nárokům na zařízení, která se starají o směrování.

Naproti tomu ve verzi 6 je k adresování použito 128 bitů a díky nim lze adresovat až 2^{128} unikátních adres. Takto velký adresní prostor disponuje natolik velkým rozsahem volných adres, že jeho vyčerpání se v nejbližší době několika desetiletí nepředpokládá. O novém protokolu IP verze 6 již vyšlo několik technických publikací, z nichž u nás nejznámější je kniha pana Pavla Straky[8], kterou lze stáhnout v elektronické podobě.

1.6.3 TCP a UDP

Jak již bylo zmíněno v podkapitole o síťových modelech, jsou tyto protokoly užity v transportní vrstvě především pro řízení toku dat a pro adresaci síťové služby v rámci počítače.

TCP zajišťuje především spolehlivé, spojově orientované spojení dvou koncových uzlů. K zajištění těchto vlastností je potřeba určitá režie, která bohužel vyžaduje určité množství výpočetních zdrojů na uzlech, které tento protokol podporují. Pro zajištění odpovídajícího zacházení se segmenty, je v záhlaví těchto segmentů přidáno celkem rozsáhlé množství informačních polí. Mezi ty nejdůležitější patří zdrojový a cílový port, sekvenční čísla zpráv, pole příznaků, aktuální velikost okna a kontrolní součet hlavičky a přenášených dat. Sekvenční čísla slouží k udržení pořadí segmentů a zároveň slouží k detekci ztráty dat. V poli příznaků jsou takové příznaky, které slouží jak pro vytváření, udržování, tak i rušení nepotřebného spojení. Nejdůležitějšími příznaky jsou SYN, který indikuje počátek navázání spojení. Jeho opakem je příznak FIN, který navázané spojení ruší. V poli příznaků je i příznak pro potvrzení, který se jmenuje ACK. Dalším důležitým polem v záhlaví segmentu je velikost okna. Hodnota v tomto poli udává maximální počet segmentů, které může být vysláno, aniž by se čekalo na potvrzení od protější strany. Posledním poměrně důležitým polem je pole kontrolního součtu, které lze použít pro kontrolu nepoškození dat a záhlaví segmentu.

V případě, že není potřeba vysoké spolehlivosti přenosu, může být použit transportní protokol UDP. Tento protokol nevytváří žádné spojení. V případě, že jsou k dispozici data, jsou okamžitě vyslány k cíli. V záhlaví tohoto protokolu proto existuje pouze pár polí, a to zdrojový a cílový port, délka segmentu a kontrolní součet. Význam těchto polí je prakticky identický jako u protokolu TCP. Nízké záhlaví je využíváno například při přenosu zvuku nebo obrazu, kdy při ztrátě malého množství dat nemusí uživatel tuto ztrátu prakticky postřehnout.

Podrobněji se problematice sítí, abstraktních modelů, síťových vrstev a protokolů na těchto vrstvách zabývá zdroj [9].

1.6.4 HTTP

Zkratka HTTP pochází z anglických slov Hypertext Transfer Protocol. Tento protokol pracuje na aplikační vrstvě modelu TCP/IP. Jeho hlavním úkolem je přenos takzvaných hypertextových dokumentů ve formátu podle standardu HTML. Tento standard byl stejně jako HTTP protokol vytvořen organizací W3C[27], která mimo jiné stojí za tvorbou dalších standardů, jako je například XHTML, což je rozšíření klasického HTML. Dále pak třeba XML, jež je obecným značkovacím jazykem pro snadnější multiplatformální výměnu dat, nebo standard CSS, který je používán pro popis vzhledu webových stránek.

Způsob komunikace je založen na modelu klient/server. Klient vytváří dotazy, které odesílá serveru. V těchto dotazech se nachází takzvaná URL adresa, která slouží k jednoznačné identifikaci hypertextového dokumentu, či jiného typu souboru. Server poté na tyto dotazy náležitě odpovídá.

Podrobněji se HTTP protokolem a jím přenášenými standardy zabývá zdroj [27] nebo [26].

1.6.5 HTTPS

Tento protokol vychází z předešlého protokolu HTTP. HTTPS navíc obsahuje prvky, které zajišťují šifrování a autentizaci. K těmto účelům se nejčastěji používá SSL a jeho standardizovaná verze TLS.

Základem SSL a TLS jsou digitální certifikáty, jejichž součástí je i podpis certifikační autority, která ručí za ověření identity vlastníka certifikátu. Při vytváření spojení si strany posílají certifikáty, a pokud vyhovují nastaveným pravidlům, je vytvořeno šifrované spojení, u kterého je ověřena identita komunikujících stran. Při ověřování identity u protokolu HTTPS není podmínkou ověření klienta, je vyžadováno ověření pouze identity serveru.

Problematikou zabezpečeného spojení se blíže zabývá zdroj [11].

1.6.6 NTP

Protokol NTP je protokol pro časovou synchronizaci zařízení, které jsou připojeny do sítě internet. Tento protokol pracuje rovněž na aplikační vrstvě a pro svoji práci

používá UTC čas. Tento protokol využívá hierarchický model, ve kterém jsou jednotlivé úrovně nazývané jako „stratum“. Existuje celkem šestnáct úrovní, a to nula až patnáct. Stratum 0 je referenční úroveň, od které jsou odvozovány všechny nižší úrovně. Stratum 1 je primární úroveň přesnosti času. Server na této úrovni musí splňovat několik kritérií, aby mohl být touto úrovní označován. Stratum 2 je sekundární úroveň přesnosti času. Servery na této úrovni musí splňovat o něco méně kritérií, avšak jejich přesnost je tím pádem menší.

Vysvětlením základních pojmů, podrobnějším popisem fungování tohoto protokolu a objasněním zajištění přesnosti synchronizace času se podrobněji zabývá zdroj [28].

2 Zabezpečený přenos dat

Pod pojmem zabezpečený přenos dat si lze představit přenos informace mezi dvěma či více účastníky s tím, že je zajištěno doručení zprávy ve tvaru, v jakém byla zpráva vyslána. Zároveň by mělo být zajištěno, že zprávu nemůže číst neoprávněná osoba, natož pak i aby tato osoba přenášenou zprávu mohla modifikovat.

2.1 Korektní přenos dat

Korektní přenos dat je přenos informací z jednoho bodu sítě do druhého, tak aby nedošlo k jejich ztrátě nebo poškození. Ztrátu lze detekovat tak, že například klient v dostatečně dlouhém časovém intervalu nedostane od serveru odpověď na svůj požadavek. V takovém případě je dosti pravděpodobné, že se ztratil buď požadavek, nebo odpověď na něj.

V případě detekce poškození dat jsou k přenášené zprávě přidány data, díky kterým je nechtěná modifikace přenášených dat detekovatelná. V případě, že se pro detekci poškození použije sofistikovaná metoda, u které je pravděpodobnost neodhalení modifikace přenášených dat dostatečně malá, lze tuto metodu současně použít i jako jednoduchou, i když málo účinnou metodu odhalení modifikace dat neoprávněnou osobou.

Takto rozšířená zpráva není zcela odolná vůči útokům. Útočník může samozřejmě kromě modifikace dat modifikovat i data pro detekci změny v přenášených datech. Pro zabezpečení proti tomuto útoku se dá použít například šifrování nebo jsou

přidávána kontrolní data, která jsou generována kromě dat ve zprávě i podle tajného klíče, který útočník nezná.

2.1.1 Kontrolní součty

Kontrolní součet je zdánlivě nadbytečná informace, která v sobě nese zakódovanou informaci o vstupních datech. Jejím smyslem je informovat o tom, zda byla vstupní data modifikována či nikoliv.

Práce s kontrolními součty funguje na principu, že je vytvořen kontrolní součet určitých dat a tento součet je přidán k těmto datům. Pokud je pak třeba zkontrolovat, zda nedošlo ke změně dat, je vytvořen nový kontrolní součet a ten je porovnán s tím, který byl přidán k vstupním datům.

Mezi kontrolní součty patří například parita, výsledek modulo operace nad vstupními daty, cyklické kontrolní součty nebo algoritmy pro tvorbu tzv. hashe neboli otisku.

Parita je logická operace, která je schopna nalézt pouze lichý počet chyb. Tato vlastnost ji deklasuje na použití v prostředí, ve kterém nedochází k chybám nebo příliš nezáleží na chybovosti systému. Pravděpodobnost nalezení chyby je v tomto případě pouze padesátiprocentní. Realizace je velmi jednoduchá a proto ji lze nalézt i v nenáročných systémech

Modulo operace jsou lepším řešením pro odhalování změn vstupních dat, ale i tak nejsou schopny odhalit chyby s dostatečně velkou přesností. Jejich realizace je složitější než parita, proto se používají jen tam, kde je jejích realizace nenáročná.

Další jmenované metody jsou daleko účinnější metody pro odhalování modifikací v datech. Bohužel s nárůstem efektivity odhalování narůstá i obtížnost jejich implementace do elektronických a informačních systémů.

2.1.2 Cyklické kontrolní součty

Cyklický redundantní součet (neboli z anglického cyclic redundancy check - CRC) je funkce, která vezme vstupní data a na základě nich a určitých logických funkcí vypočte hodnotu, která je jednoznačnou pro tuto vstupní posloupnost. Při změně jediného bitu vstupních dat pak dochází k tomu, že výstupní hodnota je s určitou pravděpodobností odlišná od té původní.

Existuje řada specifikací pro tvorbu cyklických redundantních součtů. Mezi ty nejznámější patří CRC-8, CRC-16-IBM, CRC-32-IEEE 802.3 Tyto standarty se především liší šířkou výstupní hodnoty, dělícím polynomem nebo reakcí na externí případy (blok vstupních dat obsahuje na začátku větší množství nul).

I když pravděpodobnost odhalení změny v datech je daleko vyšší než u paritního zabezpečení nebo modulo operací, stále existuje určitá pravděpodobnost toho, že cyklický redundantní součet změnu ve vstupních datech neodhalí. Pro výpočet pravděpodobnosti zjištění změny lze použít následující vztah.

$1 - 1 / 2^N$

kde N je délka stupně klíče. S čím dál větším N je pravděpodobnost nenalezení chyby menší, ale nikdy nebude nulová.

2.1.3 Hash

Hash neboli otisk je velmi účinná technika pro detekci modifikace v datech. Díky tomu, že vstupní data mnohonásobně prochází hashovacími rutinami, je pravděpodobnost nezachycení, byť jediného bitu, tak malá, že ji můžeme takřka považovat za nulovou. Rutinám, které tvoří základ těchto algoritmů se říká rundy. V těchto rundách se používá matematické operace, jako jsou posuny, negace, exklusivní OR či různé prohazování bitů zpracovávaného slova. V současnosti se používá celá řada hashovacích algoritmů, jako je například MD5, SHA, RIPEMD nebo Tiger.

MD5 je kryptograficky silný, 128bitový hashovaní algoritmus. Vychází z předešlých verzí tohoto algoritmu, kterými byly MD2 až MD4. Dříve byl tento algoritmus hojně používán, bohužel se časem podařilo provést kolizní útoky, a proto se od něho upouští pro případ nalezení nějaké závažnější chyby.

Algoritmus MD5 byl nahrazen algoritmem SHA. Tento algoritmus je daleko silnější než jeho předchůdce. Do dnešní doby na něj nebyl úspěšný kryptografický útok. Existuje několik verzí. Verze SHA1 generuje 160 bitový otisk, naproti tomu novější verze SHA2 umožňuje vygenerovat až 512 bitový otisk. V současnosti pak je připravována verze SHA3, která by měla přinést ještě silnější hashovací funkci.

Podrobněji se problematikou hashovacích algoritmů a jejich použití v reálných výpočetních systémech zabývá publikace v prameni [11].

2.2 Šifrování

Šifrování neboli kryptografie se zabývá způsobem, jak původní zprávu pozměnit tak, aby ji případný útočník nemohl číst nebo dokonce modifikovat tak, že by i po modifikaci dávala smysl. Zároveň je třeba zajistit, aby příjemce šifrované zprávy ji mohl pozměnit tak, aby dostal původní zprávu. Z těchto důvodů musí být šifra dostatečně kryptograficky silná, aby při dostatečně velkém úsilí útočníka ji nebylo možné prolomit. Šifra, která už z principu tento požadavek splňuje, je takzvaná dokonalá šifra. Jejím principem je to, že původní zpráva je zašifrována klíčem, který je minimálně tak dlouhý, jako je původní zpráva a zároveň již tento klíč nikdy nebude použit pro šifrování další zprávy. Nevýhodou této šifry je požadavek, aby všichni účastníci šifrování nebo dešifrování měli určitou zásobu šifrovacích klíčů. Toto bohužel v mnoha případech není možno dodržet, proto se používají šifry s klíčem, který je kratší než šifrovaná zpráva a tím pádem se jeho použití musí opakovat. Kryptografickou sílu těchto šifer musí kompenzovat kvalitní kryptografický algoritmus.

Základním rozdělením šifer je podle toho, zda šifrovací i dešifrovací klíč je stejný nebo zda jsou použité rozdílné klíče. V prvním případě, kdy jsou klíče pro šifrování i dešifrování stejné se tyto šifry nazývají symetrické. V druhém případě pak těmto šifrám říká asymetrické šifry.

2.2.1 Symetrické šifrovací systémy

Jelikož u těchto šifrovacích systémů jsou klíče stejné, pak každý, kdo zná tento klíč, může dešifrovat přenášené zprávy. Z těchto důvodů je třeba udržet tento tajný klíč v bezpečí. Existují dva typy symetrických systémů, které se od sebe liší tím, jakým způsobem jsou zprávy šifrovány tajným klíčem.

Prvním typem jsou proudové šifry. Ty šifrují jednotlivé bity zprávy jednotlivými bity klíče. Proudové šifry jsou velmi rychlé, ale jejích rychlost je na úkor kryptografické bezpečnosti. Mezi hojně používané proudové šifry patří například RC4 nebo A5.

Druhým typem jsou blokové šifry. Na rozdíl od proudových šifrují zprávy po blocích přesně dané délky. Šifrování probíhá pomocí několika šifrovacích bloků, přes které postupně prochází zpráva. Několikanásobným provedením šifrovacího algoritmu nad těmito procházejícími zprávy je zajištěna vyšší kryptografická bezpečnost než u proudových šifer. S tím však souvisí i rychlost šifrování, která je nižší. V současnosti se používá celá řada blokových šifer jako je AES, IDEA, DES a jeho rozšířená verze Triple DES.

Blokové šifry lze použít v různých módech, které se mezi sebou liší tím, jakým způsobem dochází k aplikaci šifrovaní bloků. Mezi ty nejznámější pak patří módy ECB, CBC, PCBC, CFB a OFB. Nejvíce používané jsou módy ECB a CBC. ECB je nejjednodušším módem, kdy vstupní data jsou šifrována v šifrovacím stroji, a na výstupu je zašifrovaný kryptogram zprávy. V módu CBC je pak výstup šifrovacího stroje přiveden na předřazený člen, jehož úkolem je provést exklusivní OR operaci mezi vstupním blokem dat a výstupním blokem dat. Podrobněji o typech módů pojednává zdroj [12].

2.2.2 Asymetrické šifrovací systémy

Asymetrické šifrovací systémy jsou založeny na principu existence dvou klíčů, veřejného a soukromého. Veřejným klíčem může kdokoliv zašifrovat zprávu, ale pouze vlastník soukromého klíče může provést opětovné dešifrování.

Základem těchto algoritmů jsou matematické problémy, jejichž obtížnost řešení je exponenciální. Využívá se matematických problémů jako je faktorizace velkých prvočísel, problémů diskrétního logaritmu nebo sčítání bodů eliptické křivky.

Asymetrické šifrovací systémy jsou mnohem pomalejší než systémy symetrické, avšak jejich kryptografická bezpečnost je mnohem vyšší. Používají se v místech, kde je třeba vysokého bezpečí a zároveň není potřeba přenášet vysoké množství dat zašifrované těmito šiframi. Těmito místy použití může být proces vyjednávání společného šifrovacího klíče, ověřování identity nebo mohou být použity při práci s digitálními podpisy.

Podrobněji se typy, vlastnostmi a principem fungování těchto šifrovacích systémů zabývá zdroj [13], kde je mimo jiné probírána i jejich kryptografická bezpečnost.

2.3 Autentizace účastníků

Ani jedna z předešlých metod primárně neřeší problematiku ověření toho, zda protistrana komunikace je tím, za koho se vydává. K ověření identity protistrany komunikace se používá autentizační metoda. Těchto autentizačních metod existuje celá řada. Mezi hojně používané patří protokoly, jako jsou Kerberos, SecurID, EAP nebo třeba starší protokoly PAP a CHAP. Každý z těchto protokolů zajišťuje určitou úroveň zabezpečení. Jejich základem bývá výměna zpráv, které bývají nejčastěji šifrovány dostatečně silným šifrovacím algoritmem. Pokud protistrana dokáže takto šifrované zprávy správně dešifrovat a reagovat na ně zprávami, které bude obsahovat správnou odpovědi na příchozí zprávy, je to dobrým předpokladem korektně provedené autentizace.

2.3.1 Digitální podpisy

Digitální podpis stejně jako normální podpis má potvrzovací charakter. Pomocí digitálního podpisu autor stvrzuje svoje stanovisko k elektronickému dokumentu. Toto stanovisko nemůže být zpochybněno a zároveň má autor absolutní kontrolu nad svým podpisem, který lze jen velmi těžko zpochybnit.

Při práci s digitálním podpisem je využito dvou operací. První je podepsání dokumentu, kdy je nejprve vytvořen elektronický otisk zprávy a poté je pomocí asymetrické šifry a soukromého klíče vytvořen podpis tohoto hashe. Takto vytvořený podpis je přidán k elektronickému dokumentu. Tímto je stvrzeno stanovisko majitele soukromého klíče s dokumentem v znění jaké bylo v okamžik podpisu.

Druhou operací, kterou při práci s digitálním podpisem může být potřeba, je ověření podpisu. Při ověřování je vytvořen hash zprávy, současně s tím je dešifrován podpis veřejným klíčem a je získán další hash zprávy. Poté jsou oba hashe porovnány a jestli se shodují, tak je podpis zprávy akceptován a zároveň je potvrzeno, že od podpisu dokumentu nedošlo k jeho neoprávněné změně.

2.3.2 Certifikáty

Slabinou digitálních podpisů je problém doručení veřejného klíče. Není totiž zaručeno, že veřejný klíč, který nám byl doručen nezabezpečenou cestou, nebyl pozměněn. Tuto slabinu proto řeší certifikáty. Jejich principem je infrastruktura PKI, ve které hlavní roly

hraje certifikační autorita. Tato autorita generuje certifikáty, které obsahují podpis této certifikační autority. Základem je pak to, že uživatel, který má certifikát a tím pádem i veřejný klíč této certifikační autority může ověřovat věrohodnost certifikátů podepsaných tímto certifikátem.

Podrobněji se fungováním digitálních podpisů, certifikátů a infrastrukturou veřejných klíčí (PKI) zabývá zdroj [13].

3 Prostředky současných počítačů

Současné počítače prošly velkým technologickým vývojem. V době svého vzniku zabíraly rozlohu velké výrobní haly. Postupem času se však díky pokrokům v technologiích změnily do podoby a rozměrů těch dnešních. Již v dobách svého zrodu znamenaly tyto počítače pro své vlastníky velkou výhodu oproti jejich konkurenci. Tato převaha je důležitá i v současnosti a proto se společnosti, organizace i jednotlivci snaží vlastnit takové výpočetní prostředky, které by jim zajistili jejich úspěch a prosperitu.

Pro různé účely a řešení problémů existuje mnoho typů těchto výpočetních systémů, které se liší způsobem, jakým pracují, jaká vstupní data umožňuje zpracovávat a jaká data lze na jejich výstupu dostat.

Nejzákladnějším rozdělením počítačů je, zda jejich principem činnosti je zpracovávání analogových nebo číslicových dat. Z tohoto rozdělení jsou odvozeny i počítače a to buď analogové počítače, které zpracovávají analogová data nebo číslicové počítače, které umožňují zpracovávat digitální data. Analogové počítače se v současnosti skoro nepoužívá, až na několik málo speciálních případů nasazení. Zcela výhradně se využívají číslicové počítače, které ve svých funkčních blocích pracují jen s diskrétními hodnotami. V případě, že je třeba pracovat s nediskrétními hodnotami, je nutno provést odpovídající převod.

3.1 Architektury

V době, kdy vznikaly první počítače, bylo třeba vytvořit několik konceptů, podle kterých by bylo možné vytvářet tyto počítače. Proto vzniklo několik konceptů, z nichž se masivněji rozšířily pouze dva.

Prvním konceptem je Neumannovo schéma počítače, jehož základními principy je práce ve dvojkové soustavě. Programy a data se v operační paměti nacházejí ve společné paměti, výpočetní jednotka může přistupovat přímo do paměti, která pracuje na stejné rychlosti jako výpočetní jednotka. Posledním principem Von Neumannova počítače je výkonné výpočetní jádro, které je realizováno aritmeticko-logickou jednotkou, která velmi rychle provádí základní matematické operace.

Druhým hojně používaným konceptem je Harvardská architektura, která používá podobných prvků jako Von Neumannova architektura. Oproti Von Neumannova schématu počítače má tato architektura dvě od sebe oddělené paměti. Jednu pro data a druhou pro program. Díky tomuto rozdělení je nutno znát poměr mezi množstvím dat a instrukcí už v době návrhu programu. Poměrně často má každá z těchto pamětí vlastní sběrnici, po které se přenáší buď data, nebo instrukce. Tímto navýšením počtu sběrnic rovněž roste i výkonnost celého systému. V dalších parametrech se tato architektura příliš neliší od Von Neumannovy architektury. Tato architektura se často používá u digitálních signálových procesorů nebo třeba u malých jednoúčelových mikrokontrolérů.

3.1.1 Intel x86

Současné počítače využívají výhod obou předešlých architektur, a proto se v dnešní době můžeme setkat se systémy, které v sobě aplikují prvky obou těchto architektur. Příkladem mohou být dnešní procesory rodiny Intel x86, které ve svém jádře využívají prvky harvardské architektury, ale navenek se chovají jako procesory vycházející z Von Neumannovy architektury.

Počítače založené na procesorech rodiny Intel x86 mají na poli osobních počítačů dominantní postavení, které si vydobyly svojí jednoduchostí, cenou a možností pořídit jednotlivé jeho komponenty od různých dodavatelů. Jádrem celého počítače je v tomto případě procesor, který je zpětně kompatibilní k původnímu osmibitovému procesoru Intel 8086. Tento procesor vychází, jak již bylo uvedeno v předešlém odstavci, z Von Neumannova konceptu počítače. Proto má tento počítač pouze jednu paměť,

která slouží jak pro ukládání dat, tak i instrukcí. Tímto je zaručena určitá univerzálnost použití těchto počítačů.

K tomuto procesoru jsou přes podpůrné obvody připojeny periferie a rozhraní různých určení a typů. Jejich úkolem je umožnit vstup povelů uživatele, výstup výsledků podnětů uživatele na zobrazovací zařízení nebo umožnit vstupně výstupní operace s periferním zařízením či dalším počítačem nebo počítači.

Podrobněji se problematikou architektur počítačů, v dnešní době hojně používanými procesory, typy operačních pamětí a grafických zařízení zabývá literatura [10].

3.2 Softwarové vybavení

S rozvojem technického vybavení souvisí i vývoj jeho aplikačního vybavení, které využívá jeho možnosti. V dávných dobách prvních sálových počítačů se aplikační vybavení tvořilo přímo ve strojovém kódu. Tvorba tímto způsobem byla velmi obtížná a dosti často docházelo k chybám v aplikacích. Z těchto důvodů vznikl jazyk symbolických adres, který přinesl snazší a přehlednější tvorbu těchto aplikací. I přesto však vývoj rozsáhlejších aplikací byl náročný, proto vznikly vyšší programovací jazyky.

S opakovaným vývojem podobných aplikací si programátoři uvědomovali možnost znovupoužití určitých částí zdrojového kódu. Proto začaly vznikat knihovny funkcí, které nabízí vývojářům rychlejší vývoj, jelikož už někdo před nimi obdobnou funkci potřeboval a proto ji implementoval způsobem, který ji umožňuje znovu použít v nejrůznorodějších případech. Postupem času začaly vznikat komplexnější knihovny, takzvané frameworky, které mimo funkcí mohou obsahovat i podpůrné programy, implementace návrhových vzorů nebo takzvané wrappery, které umožňují snazší přístup ke knihovnám nebo rovnou vytváří standardizovaný přístup k určitému portfoliu knihoven, z nichž každá má jiné rozhraní pro komunikaci.

I přes to, že vznikaly nové programovací jazyky, které nabízely vývojářům nové a nové prvky, jejichž úkolem bylo zpřehlednit a nabídnout intuitivní způsob programování, bylo třeba vytvořit nové metodiky tvorby aplikací. K původnímu strukturovanému programování přibyly metodiky, jako jsou objektově orientované programování, agilní programování nebo službami orientované programování.

Kromě programovacích jazyků, které nativně běží na technickém vybavení, postupem času vznikly i programovací jazyky, které ke svému spuštění potřebují takzvaný interpret, který vykovává funkce podle příkazů nalezených ve zdrojovém kódu. Speciálním případem jsou pak jazyky, které se pouze částečně přeloží do takzvaného bytekódu, který je přeložen až při spuštění skriptu v interpretu tohoto programovacího jazyka.

3.3 Socket

Socket je abstraktní datová struktura, která slouží v aplikaci jako přístupový objekt pro komunikaci v síti. Každý socket lze jednoznačně identifikovat pomocí dvou hodnot, a to adresou IP a cílovým portem. Existují dva druhy socketů. Prvním je spojově orientovaný, který zajišťuje obousměrnou komunikaci neomezeného množství dat. Druhým je nespojově orientovaným socket, který slouží k odeslání nebo přijmutí určitého konečného množství dat v rámci jednotlivých zpráv.

Jako spojově orientovaný socket lze uvést socket typu TCP, který využívá transportního protokolu TCP. Před práci s tímto typem socketu musí být tento socket inicializován. Nejčastěji se k tomu užívá funkce *socket(),* která přichystá socket pro práci se síťovými službami. Teprve poté se může provádět navazování spojení, které je na klientské straně a na serverové straně odlišné. Na straně klienta se nejčastěji zavolá pouze funkce *connect(),* kdežto na straně serveru se musí provést svázání socketu s názvem nejčastěji pomocí funkce *bind()* a poté se socket uvede do stavu vyčkávání na příchozí spojení, nejčastěji pomocí funkce *listen().* V okamžik, kdy je žádost na spojení, je tento požadavek obsloužen funkcí *accept()*. V tomto okamžiku lze data odesílat a přijímat zápisem nebo čtením ze socketu a v případě, že již nebude třeba tohoto socketu, lze ho uzavřít, což se provádí na straně serveru nebo klienta nejčastěji funkcí *close()*.

U nespojově orientovaného spojení, u kterého se využívá transportního protokolu UDP, je situace odlišná. Stejně jako v předešlém případě se musí socket inicializovat například funkcí *socket()*. Po této inicializaci již lze tento socket využívat.

U obou typů těchto socketů lze vytvořit asynchronní přístup k takto vytvořenému a inicializovanému socketu. Při tomto přístupu se nemusí periodicky testovat, zda data došla, ale lze provádět jinou činnost, a v okamžiku, kdy jsou data přijata, se vyvolá

procedura, která zajistí zpracování přijatých dat. U spojově orientovaných lze asynchronně přijímat i přicházející spojení. Výhodou tohoto přístupu je ušetření výpočetního výkonu, který by jinak byl použit na nekonečné testování příchodu zprávy nebo spojení.

Prací se síťovými protokoly nebo sockety se zabývá zdroj [14], ve kterém lze nalézt i potřebnou teorii.

3.4 .NET Framework

.NET Framework je platforma od společnosti Microsoft, která má sloužit k ulehčení, urychlení a zefektivnění vývoje aplikací pro operační systémy Windows na osobních, serverových a mobilních počítačích. Na trhu existuje několik podobných platforem, ale nejznámější je platforma JAVA. Ta je na rozdíl od .NET Frameworku více multiplatformální. Avšak za cenu toho, že neimplementuje funkce specifické pro dané platformy na kterých je implementována. .NET Framework je určen pouze pro operační systémy Microsoft Windows a proto má možnost implementovat i úzce vázané funkce pro daný operační systém. Dalším významným rozdílem mezi platformou Java a .NET je možnost použití programovacích jazyků. Platforma Java umožňuje použít pouze jeden programovací jazyk, který se jmenuje stejně jako celá platforma. Oproti tomu .NET nabízí více programovacích jazyků jako je Managed C++, C# (vyslovované anglicky jako C Sharp), J# (jazyk blízký jazyku Java), Visual Basic a několik dalších méně používaných jazyků.

Podrobněji se o frameworku .NET zabývají zdroje [15] a [16], ve kterých lze získat jak všeobecné tak i detailnější informace. Zároveň jsou zde odkazy na technologie, které tohoto frameworku využívají, nebo lze v těchto zdrojích nalézt odkazy na aktuální verze těchto frameworků.

3.4.1 Vlákna

Moderní operační systémy umožňují spouštět více úloh současně, a to i když mají k dispozici jednu výpočetní jednotku. Toho dosahují rychlým přepínáním mezi jednotlivými procesy. Tomuto procesu se říká multitasking. Pokud existuje v rámci procesu více vláken, je toto přepínání označováno jako multithreading. Existuje-li

v jednom počítači více výpočetních jednotek, může na každém z nich běžet jeden proces nebo vlákno.

Rozdíl mezi vláknem a procesem je takový, že v rámci jednoho procesu může existovat více vláken, které mohou mít různé úkoly. Proto se dosti často setkáváme s operačními systémy, které umožňují spíš běh vláken než běh pouze samostatných procesů. Tímto je zajištěno to, že výpočet složité matematické úlohy může být rozdělena na vlákna a každé z těchto vláken pak může běžet na samostatné výpočetní jednotce počítače.

Prakticky každá úloha potřebuje komunikovat s dalšími úlohami, proto je potřeba zajistit, aby jedna úloha nemohla ovlivnit běh druhé a naopak. Často také úlohy používají jednu společnou paměť pro sdílení informací nebo řízení běhu. V tomto společném místě může dojít ke kolizím, kdy více vláken se snaží například modifikovat data. K odstranění těchto míst se používá mnoho metod, jako jsou například mutexy, fronty nebo zámky.

3.4.2 Události

Jsou v počítačové technice stavy, které nepřichází v pravidelných intervalech, ale náhodně. Po detekci tohoto stavu je nejčastěji volána funkce, která má za úkol nějak na tento stav reagovat. Dosti se události podobají přerušení, avšak na rozdíl od přerušení nedochází nutně k přerušení hlavní větve programu.

3.4.3 Časovač

Je zvláštním typem události, kdy je její příchod přibližně v pravidelných intervalech. Často je časovač řešen tak, že jsou v rámci počítače počítány jednotky času. Pokud uběhne odpovídající počet těchto jednotek, je vyvolána událost, která je obsloužena v nějaké funkci. V okamžiku, kdy je vyvolána událost, je znovu nastaven časovač a vše se znovu opakuje.

3.4.4 Šifrování

Knihovny s algoritmy pro šifrování ve frameworku .NET jsou stejně jako u nativně spouštěných aplikací přístupné pomocí takzvaných providerů. Jejich smyslem je existence jednotné implementace šifrovacích algoritmů dostupná přímo v operačním systému. Kromě šifrovacích algoritmů poskytují také funkce určené pro vytvoření hashe.

Mezi šifrovací algoritmy, které jsou v systému pomocí providerů přístupné, patří například DES, RC2, RSA nebo AES. Některé jmenované algoritmy jsou dostupné až od určité verze. Například posledně jmenovaný algoritmus AES je přístupný až od verze frameworku .NET číslo 3.5.

Provideři dále poskytují hashovací funkce jako jsou dnes hojně používané MD5 nebo SHA1 a SHA2.

3.4.5 ActiveX

ActiveX je jakýmsi frameworkem pro znovu použitelný kód. Jeho hlavním úkolem je zpřístupnit objekty typu COM a OLE, které jsou uloženy v operačním systému. Tato technologie je prakticky výhradně spjata s operačním systémem Microsoft Windows. ActiveX není přímo součástí .NET Frameworku, ale lze v něm tuto technologii využívat.

3.4.6 XML

Je obecný značkovací jazyk, který byl vytvořen konsorciem W3C. Slouží jako univerzální prostředek pro předávání dat mezi různorodými informačními systémy. Skládá se z několika částí. První a nejdůležitější jsou elementy, které tvoří celkovou stromovou strukturu. Mezi odpovídajícími páry značek se poté nalézá obsah. V prvním z páru značek pak lze definovat atributy tohoto páru značek. Tento značkovací jazyk se rozšířil především v systémech, které mají co dočinění s webovými službami.

Podrobněji se frameworkem .NET a technologiemi v něm použitými zabývají stránky Microsoftu [17].

3.5 Virtuální zařízení

Virtuální zařízení lze definovat jako zařízení, které není z hlediska operačního systému přítomno fyzicky, ale je programově realizováno tak, aby z pohledu aplikace nebo uživatele se jevilo jako skutečné fyzické zařízení, které je připojeno k počítači.

Aplikace, která pak toto virtuální zařízení využívá, nepřistupuje k fyzickému zařízení, ale přistupuje k datovým strukturám, které leží v operační paměti a přísluší virtuálnímu zařízení.

3.6 UML

UML je univerzální jazyk pro vizuální modelování systémů. V tomto modelu lze vizualizovat buď strukturu, interakce nebo chování. Toto základní rozdělení modelů lze dále ještě rozdělit na další různé podtypy vizualizace modelů. Asi nejčastěji užívaným je diagram tříd. Dalšími hojně využívanými vizualizacemi jsou například diagramy užití, stavové a sekvenční diagramy a diagramy komponent.

Diagram tříd slouží pro vizualizaci tříd objektů a vztahů mezi nimi. Třída přitom může obsahovat atributy a metody, které mají určitou dostupnost. Těmito dostupnostmi mohou být *private*, což znamená nedostupnost pro ostatní objekty. Dále pak *protected*, což znamená přístup jen ze tříd, které tuto třídu dědí. Posledním stavem dostupnosti může být *public*, který udává dostupnost pro všechny objekty.

Mezi třídami objektů může existovat několik typů vazeb, jako jsou například asociace, dědičnost, realizace, agregace a kompozice. Prvně jmenovaná znázorňuje pouze obecný vztah mezi jednotlivými třídami. Dědičnost je vztah, při kterém potomek dědí po rodičovské třídě veřejné a chráněné typy atributů a metod. Třetím vztahem je realizace, která určuje, které metody musí dědící třída implementovat. Poslední dvě jmenované jsou znázorněním toho, že jedna třída obsahuje objekt té druhé. Rozdíl mezi agregací a kompozicí je takový, že vztah kompozice definuje daleko užší vztah dvou tříd než agregace.

Podrobněji se UML zabývá zdroj [18], ve kterém je daleko podrobněji popsáno a rozebráno použití různých typů diagramů.

4 Modul Rabbit RCM3700 a vývojový KIT

Výrobky od společnosti Rabbit Semiconductor byly vybrány pro realizaci diplomové práce z důvodu dostupnosti kvalitní dokumentace a rozsáhlého portfolia knihoven i pro složitější účely, jako je implementace šifrovacího algoritmu AES nebo implementace vlastního operačního systému. Kromě toho k dalším výhodám výrobků tohoto výrobce patří velká paleta řídicích modulů a patřičného příslušenství. Jádrem řídicích modulů jsou mikroprocesory, které jsou osmi nebo šestnáctibitové. Dále na těchto modulech bývá paměť flash pro program, paměť SRAM jako operační paměť

mikroprocesoru, externí flash paměť pro data a samozřejmě externí periferie jako jsou Wi-Fi, ZigBee, RS-232 nebo Ethernet.

Všechny tyto moduly lze programovat a ladit pomocí jednotného vývojového prostředí, které využívá jako programovací jazyk Dynamic C. Tento programovací jazyk vychází z jazyka C, avšak nedodržuje striktně všechny jeho syntaktická a sémantická pravidla. Dynamic C rozšiřuje základní verzi jazyka C o několik programovacích technik a přidává podporu, kterou v konzervativním jazyce C nelze nalézt.

Existuje několik verzí tohoto vývojového prostředí. Pro námi použitý vývojový KIT jsou k dispozici verze řady 9. Tato řada přímo disponuje nebo lze rozšířit o knihovny, které jsou potřeba pro realizaci diplomové práce. Bohužel však tato řada již není hlavní vývojovou větví a od toho se odvíjí i dostupnost nejnovějších aktualizací knihoven, které pro tuto řadu nejsou dostupné. Jistým řešením by mohlo být použití hlavní vývojové verze, která v době psaní diplomové práce nese číslo vývojové řady 10. Tato verze však není použitelná s námi použitým řídícím modulem.

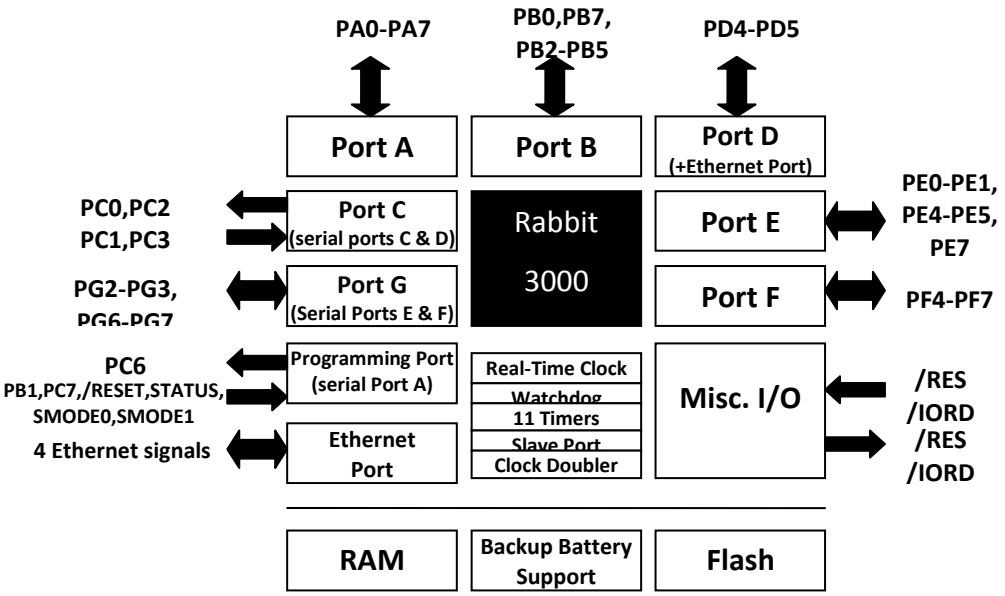
Kompletní nabídku hardwarových i softwarových produktů lze nalézt ve zdroji [19].

4.1 Řídicí modul Rabbit RCM3700

Jádrem tohoto řídicího modulu je mikroprocesor Rabbit 3000A, který pracuje na frekvenci hodinového signálu 22.1 MHz. Tento mikroprocesor je osmibitovým mikroprocesorem a jeho jednotlivé bloky lze vidět na následujícím obrázku číslo tři, který byl přejat z dokumentace výrobce modulu RCM3700 [20].

Jak je vidět, tento mikroprocesor obsahuje bloky, které lze najít v běžných mikrokontrolérech známých světových výrobců. Mimo to obsahuje hodiny reálného času, násobič hodinového signálu nebo porty pro řízení komunikace s čipem Ethernetu. Externě tento mikroprocesor pak může přistupovat k operační paměti nebo paměti flash. Velikost paměti, kterou může tento mikroprocesor adresovat je 2^{19}, což odpovídá paměti o velikosti 512kB. Kromě toho je na tomto modulu k dispozici až 1MB sériové flash paměti pro ukládání uživatelských dat. Co se týče sérových portů, tento mikroprocesor disponuje pěti nezávislými kanály portu podle upraveného standardu RS-232. Některé z těchto portů však jsou pro speciální účely jako

je například Serial Port A, který slouží pro programování a ladění, avšak lze použít i pro běžnou komunikaci.



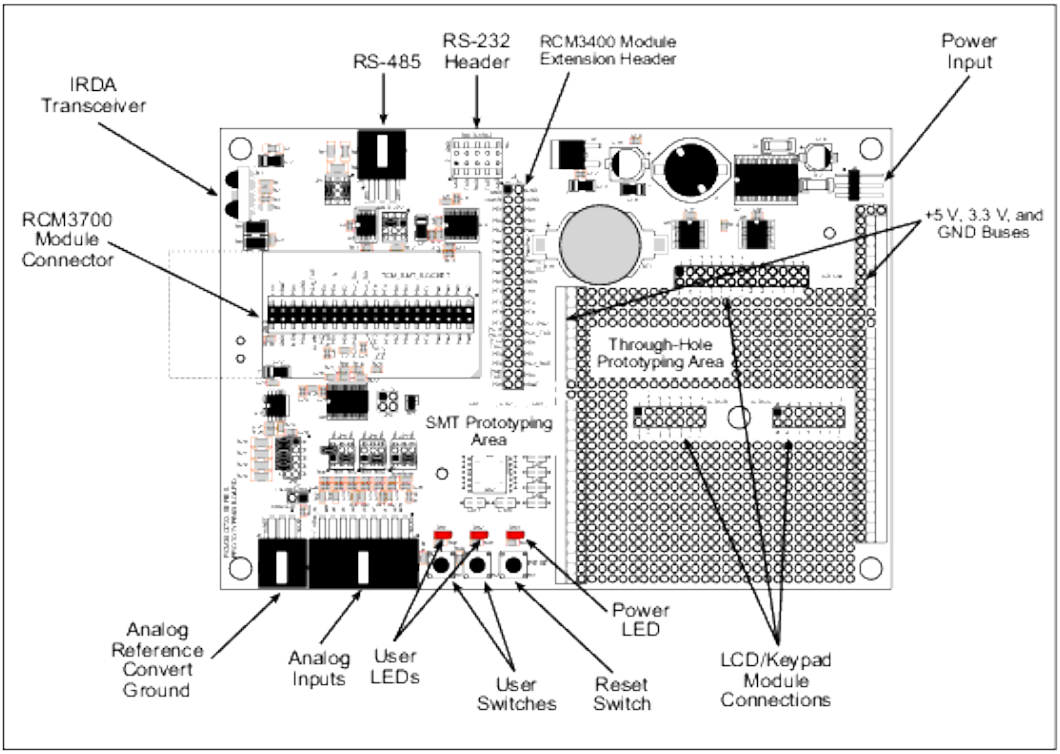
Obrázek 3: Části mikroprocesoru Rabbit 3000

Další parametry tohoto řídícího modulu lze nalézt v prameni [20], kde jsou mimo hardwarové specifikace přístupné i softwarové techniky pro práci s tímto mikroprocesorem a jeho periferiemi. Kromě toho obsahuje bližší seznámení s Ethernet portem, který je na tomto modulu umístěn.

4.2 Vývojový KIT

Řídicí modul RCM 3700 lze použít ve dvou prototypových deskách. První je označena jako vývojová deska RCM3700 a disponuje zdrojem pro tuto vývojovou desku a řídicí modul, převodníky A/D a dále převodníky úrovní sérového portu s úrovněmi CMOS 3.3V na odpovídající úroveň podle standardu RS-232 nebo RS-485. Mimo to pak umožňuje připojit k sériovým portům E a F vysilač a přijímač IRDA. Druhá prototypová deska nese označení RCM3720. Tato deska disponuje pouze napájecími obvody a obvody pro změnu úrovní podle standardu RS-232. Obě vývojové desky pak disponují rozhraním pro připojení externího modulu.

Obě vývojové desky jsou detailně popsány ve zdroji [20], ve kterém lze nalézt následující obrázek rozmístění funkčních částí na vývojové desce RCM3700.



Obrázek 4: Prototypová deska RCM3700

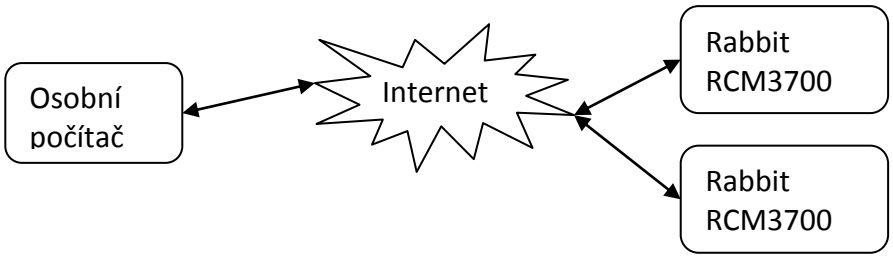
Pro realizaci diplomové práce byl použit celý vývojový KIT, který obsahuje tuto prototypovou desku včetně řídicího modulu, programovacího kabelu, externího převodníku RS-485, odpovídajících propojovacích a napájecích kabelů včetně zdroje a softwarový balíček, který obsahuje vývojové prostředí, knihovny pro práci s Ethernetem a sériovými porty. Bližší informace o tomto KITu lze získat ve zdroji [21].

5 Návrh celého konceptu

Před praktickou realizací musel být proveden návrh celého konceptu, který zahrnuje popis fungování dílčích bloku celého systému a jeho interakci s ostatními bloky. Teprve na základě pečlivého návrhu šlo provést dostatečně rychlou a kvalitní realizaci požadovaného komunikačního systému.

Základním komunikačním případem je komunikace mezi vývojovým KITem a aplikací běžící na osobním počítači tak, aby bylo možno tímto vytvořeným spojením například odečítat údaje ze vzdáleného zařízení nebo naopak toto zařízení vzdáleně řídit. Tato komunikace musí probíhat tak, aby nebyla prozrazena přenášená data a zároveň

aby byly informace přenesené bez jejich ztráty nebo poškození. Autenticita komunikujících stran musí být ověřena dostatečně silným autentizačním algoritmem odolným vůči útokům na tento algoritmus. Zároveň musí být celý koncept řešen s ohledem na možnost záměny šifrovacího nebo autentizačního algoritmu za jiný. Nejobecnější schéma tohoto konceptu je znázorněno na následujícím obrázku, ve kterém figuruje jeden počítač a dva moduly Rabbit RCM3700. V rámci počítače je možno vytvořit několik přenosových kanálů, které mohou směřovat buď k jednomu, nebo druhému modulu, přičemž v rámci jednoho modulu Rabbit může existovat více přenosových kanálů stejně jako na osobním počítači.

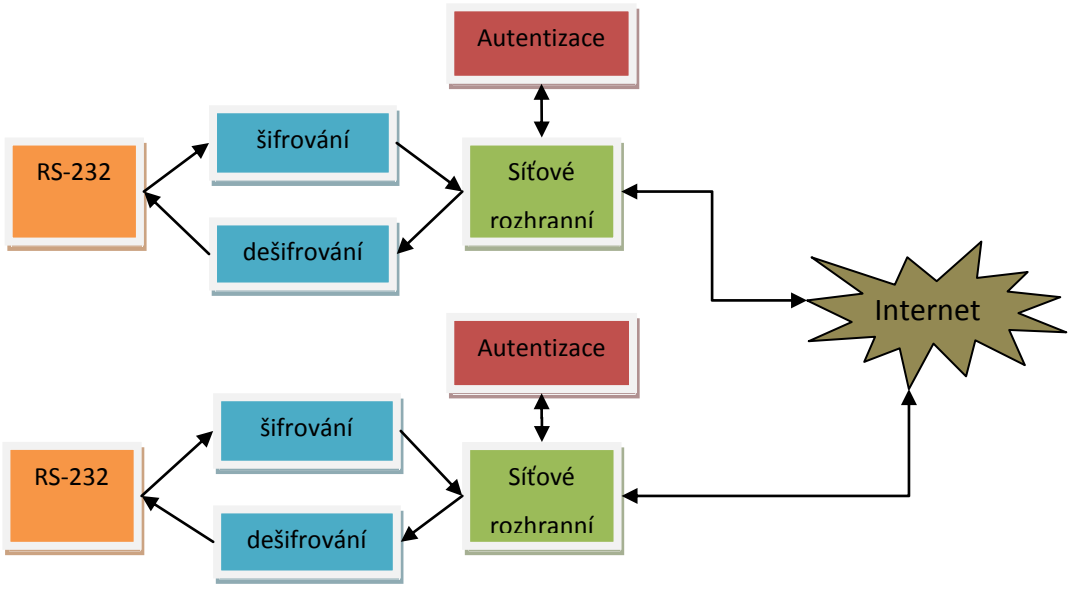


Obrázek 5: Funkční bloky celého konceptu

5.1 Funkční bloky

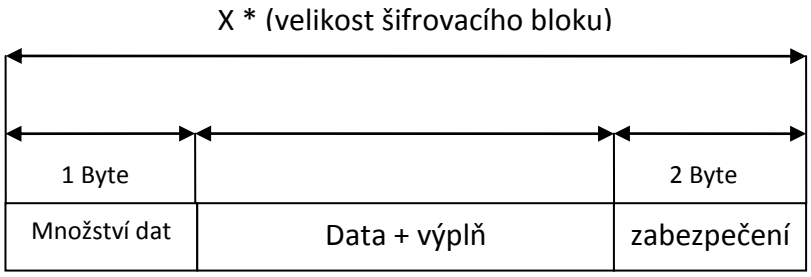
Základem celého konceptu je existence přenosového kanálu, ve kterém se data přenáší podle předem daného protokolu. Celý přenosový kanál lze rozdělit na několik funkčních částí, které mají svůj účel. Jednotlivé bloky lze vidět na následujícím obrázku. V jediné instanci aplikace na osobním počítači nebo v jediném KITu Rabbit je možnost vytvořit více těchto kanálů, které se vzájemně nemohou ovlivňovat. Součásti jediného kanálu lze vidět na následujícím obrázku číslo 6.

Blok RS-232 se stará o komunikaci se sériovým portem. Pokud detekuje, že na sériovou linku přišla data, načte je a předá dalšímu funkčnímu bloku pro šifrování. Naopak pokud mu předešlý funkční blok pro dešifrování předá data, musí je být schopen zpracovat a umístit na sériovou linku. V obou směrech přenosu se pracuje se surovými daty, proto neexistuje žádné záhlaví ani zápatí. Tento blok musí být schopen se připojit na sériový port nebo ho vytvořit jako virtuální zařízení.



Obrázek 6: Jednotlivé bloky celého kanálu

Dalším funkčním blokem je šifrovací a dešifrovací blok. Kromě kryptografických operací musí být tyto bloky schopny pracovat s předem danou strukturou dat. Vstupem šifrovacího bloku jsou surová data a na jeho výstupu je zašifrovaná struktura, která je na obrázku 7. Stejná zašifrovaná struktura dat je požadována na vstupu dešifrovacího bloku. Nejprve je tato struktura dešifrována, poté je zbavena záhlaví a zápatí. Výsledný blok dat je předán bloku RS-232 k dalšímu zpracování.



Obrázek 7: Struktura dat zpracovávaných šifrovacími bloky

Struktura dat, se kterými se pracuje mezi šifrovacím a dešifrovacím blokem je přesně daná. Celková velikost této struktury musí být zarovnána na násobek velikosti šifrovacího bloku. Z těchto důvodů se musí dynamicky data doplňovat výplní. Jelikož množství přenášených dat nelze žádným možným způsobem zjistit, má tato struktura

rovněž pole, které udává množství dat v poli pro data a výplň. Zároveň se musí počítat s velikostí a zabezpečením pomocí kontrolních součtů nebo hashovací funkce. Smyslem zabezpečení je zajistit odchycení špatně dešifrovaných bloků. Jelikož korektní přenos zajišťuje spolehlivý transportní protokol, může být za určitých okolností toto pole vynecháno. V diplomové práci však z testovacích a ladících důvodů byl jako zabezpečení použit algoritmus kontrolního součtu CRC-16 TTICC, který v této struktuře zabírá 2 byte. Pro účely šifrování byl jako šifrovací algoritmus vybrán algoritmus AES pro svoji kryptografickou bezpečnost, nenáročnou implementaci i do jednodušších systémů s nižším výpočetním výkonem a pro dostupnost tohoto algoritmu ve vývojovém prostředí Dynamic C.

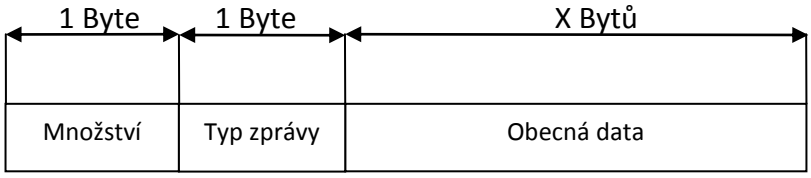
Blok autentizace je využit pouze v okamžiku hned po navazování spojení, kdy je potřeba ověřit identitu protější strany. Autentizační bloky mezi sebou posílají zprávy, které mají svůj specifický význam v rámci konkrétní autentizace. Velikost těchto zpráv se může měnit, stejně jako význam polí v jednotlivých krocích autentizace. Celý systém byl navržen tak, aby na místě autentizace mohl být jakýkoliv autentizační algoritmus.

Pro účely diplomové práce byl zvolen mírně upravený autentizační algoritmus podle zdroje [22]. Autentizační algoritmus použitý v diplomové práci je podrobněji rozebrán v následující podkapitole 5.1.1., ve které jsou detailněji popsány posloupnosti jednotlivých autentizačních zpráv.

Autentizační data jsou nejprve opatřena odpovídajícím záhlavím a teprve poté je lze odeslat. Toto se děje v bloku síťového rozhranní. Záhlaví má strukturu podle obrázku 8 a jeho podrobnější popis je uveden níže. Autentizační data nemají obecně definovanou strukturu. V rámci použitého algoritmu však platí, že prvním bytem zprávy je jednobytová identifikace této zprávy, která má zabezpečit detekci správného pořadí zpráv a tím i její korektnost. Pokud by některá zpráva přišla mimo pořadí nebo by měla nesprávnou velikost, je to bráno jako útok na autentizační algoritmus a to bude mít za následek okamžité ukončení spojení.

Posledním blokem je síťové rozhraní, které kromě navazování, udržování a ukončování spojení má daleko více funkcí. Jeho dalším hlavním úkolem je z proudu dat vytvořit zprávy a tyto zprávy poté roztřídit podle toho, zda se jedná o datové, autentizační

nebo servisní zprávy. Při odesílání dat na síť musí blok síťového rozhraní naopak vysílaná data opatřit požadovaným záhlavím. Strukturu tohoto záhlaví je vidět na následujícím obrázku.



Obrázek 8:Struktura zprávy, která je přenášená po síti

Jako obecná data lze v tomto případě přenášet buď data autentizace, užitečná data ze sériových portů, která mají strukturu podle obrázku 7 nebo třeba data při navazování spojení, která v sobě nesou informaci o nastavení serverové strany komunikace. Typem zprávy se rozumí druh dat v poli „Obecná data“. Pole množství dat pak obsahuje informaci o velikosti této struktury, která v sobě započítává i velikost tohoto pole. Obě tato pole zabírají ve zprávě přesně jeden byte. V následující tabulce je soupis všech typů zpráv použitých v diplomové práci. V případě, že bude potřeba přidat další typ zprávy, lze ji přidat jednoduchou úpravou zdrojových kódů.

Tabulka 2: Typ zprávy přijímané/vysílané na síťovém rozhraní

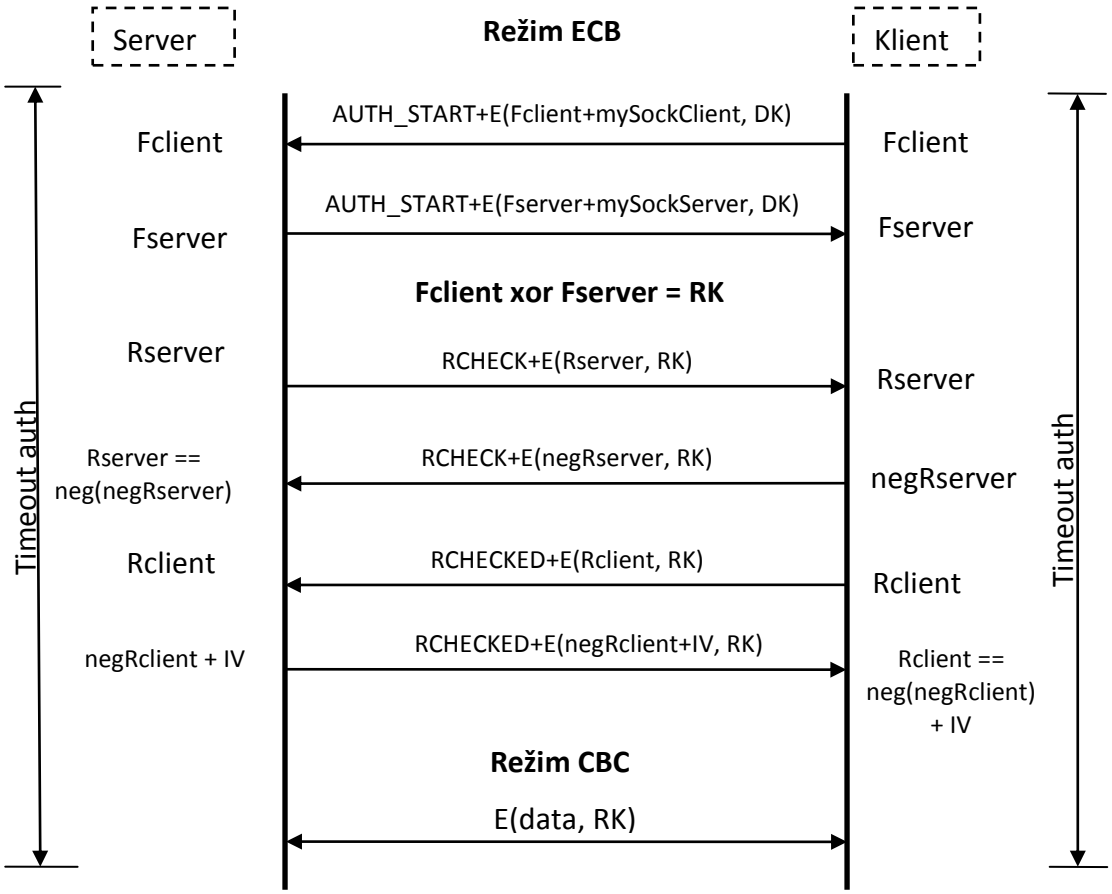
Význam zprávy	Hodnota
Užitečná data	**0x10**
Požadavek na konfiguraci kanálu	**0x20**
Odpověď na požadavek konfigurace kanálu	**0x21**
Autentizace	**0x30**

Cílem diplomové práce projektu bylo vytvořit vícekanálový systém. Z těchto důvodů bylo potřeba vytvořit knihovní funkci, která by byla schopná vytvářet jednotlivé přenosové kanály podle vstupních parametrů. V tomto případě musela být vyřešena koliznost situací, kdy jsou použity stejné parametry. Těmito parametry jsou čísla síťových nebo sériových portů. Tato koliznost byla v aplikaci na straně osobního počítače řešena pomocí varovných nebo chybových hlášení, které jsou zobrazovány

v případech, kdy jeden nebo více přenosových kanálů přistupuje k fyzickému zdroji, který je již využíván. Stejným způsobem bylo toto řešeno i ve vývojovém prostředí. Zde však tato koliznost byla řešena výpisem chybových hlášek do konzole. Pokud konzole není připojena, je tato kolize zjistitelná pouze nefunkčností přenosových kanálů.

5.1.1 Použitá autentizační metoda

Funkční diagram na následujícím obrázku ukazuje autentizační metodu, která byla použita v diplomové práci. Tento diagram zobrazuje správnou posloupnost, kterou musí obě strany absolvovat pro ověření identity protistrany. Zároveň musí během tohoto procesu být ustanoven společný náhodně generovaný klíč a inicializační vektor.



Obrázek 9: Autentizace

Struktura zpráv, které jsou mezi serverem posílány, jak bylo dříve zmíněno, není přesně daná, avšak prvním bytem je jedna z hodnot konstant AUTH_START, RCHECK nebo

RCHECKED. Účelem tohoto pole je zamezit útoku na autentizační algoritmus. Každá strana si udržuje přehled o přijatých zprávách pomocí stavů, které mají svoji přesnou posloupnost. Při přijetí zprávy mimo pořadí je autentizace považována za nezdařenou, je provedeno odpojení klienta a do logovacích mechanizmů je uložena informace o této skutečnosti. Podobným způsobem je reagováno na zprávu, která nemá odpovídající velikost.

Autentizační algoritmus je spuštěn v okamžiku, kdy klient pošle zprávu typu AUTH_START, která v sobě nese zašifrované klíčem DK náhodně vygenerované hodnoty a socket. Stejným postupem reaguje i server. Pokud by se případný útočník snažil o útok zrcadlením zpráv, je to detekováno pomocí polí mySocketClient nebo mySocketClient nesoucí v sobě informaci o použitém socketu, který je v síti internet unikátní. Existuje určitá teoretická šance, že klient a server budou používat stejný port, avšak druhá hodnota socketu veřejná IP adresa je v síti jedinečná.

Pokud má klient i server stejný klíč DK, je vytvořen klíč RK, který je pro oba stejný. Pro určení, zda mají oba stejný klíč RK a tím i DK, slouží další přenosy náhodně generovaných hodnot a jejich negací. Nejprve server vygeneruje náhodnou hodnotu *Rserver*, tu zašifruje klíčem RK a odešle klientovy. Úkolem klienta je dešifrovat hodnotu Rserver klíčem RK, znegovat, znovu zašifrovat a odeslat serveru. Server tuto hodnotu dešifruje, podruhé zneguje a porovná s uloženou hodnotou. Pokud si jsou hodnoty rovny, server ví, že klient používá stejný RK klíč a tím má i stejný klíč DK. Stejně postupuje i klient. V posledním přenosu je klientu zaslán inicializační vektor, který je použit při CBC módu šifrování užitečných dat.

Tento autentizační mechanizmus splňuje základní prvky bezpečného autentizačního algoritmu. Při každé relaci je vytvořen vždy jiný klíč RK, který zůstane utajen před případnými útočníky. Je odolný vůči útoku zrcadlení a jsou v něm aplikovány mechanizmy, které ho chrání před různými útoky.

5.2 Návrh aplikace pro osobní počítač

Pro realizaci aplikace, která bude běžet na operačním systému Microsoft Windows, byla zvolena platforma .NET Framework, pro svoje rozšíření, dostupnost potřebných knihoven a propracovanou dokumentaci.

Donedávna hojně používaný sériový port podle standardu RS-232 je v současné době nahrazován modernějším a rychlejším sériovým portem podle standardu USB [23]. Proto je v případě potřeby přístupnosti sériového portu RS-232 nutno vytvořit virtuální zařízení, které toto rozhraní počítače emuluje. K vytváření tohoto virtuálního zařízení byla vybrána komponenta od společnosti Eltima s názvem Virtual Seriál Port ActiveX[24]. Tato komponenta je přístupná pomocí platformy ActiveX, která umožňuje dynamicky vytvářet a rušit virtuální sériové porty. K takto vytvořeným portům lze přistupovat standardně jako k jakémukoliv fyzickému sériovému portu.

Jak již bylo uvedeno výše, pro šifrování byl vybrán šifrovací algoritmus AES. Ve frameworku .NET je tento algoritmus přístupný pomocí providera, který je implementován jako třída *AesCryptoServiceProvider*. Tato třída na vstupu i na výstupu umí pracovat pouze s proudem dat, který musí vycházet z třídy *Stream.* Dále pak umí šifrovat a dešifrovat pouze bloky o velikost 128 bitů. Z těchto důvodů se musí zarovnávat vstupní data na násobek této velikosti. Tato šifra je přístupná pouze ve verzi .NET Framework 3.5 a vyšších, proto se před spuštěním aplikace musí ověřit, zda je tato verze v systému dostupná.

Vývojové prostředí pro řídicí jednotku Rabbit RCM3700 disponuje knihovnami pro práci se síťovým protokolem ve verzi 4. Z těchto důvodů byl vybrán právě tento síťový protokol, jelikož vlastní implementace verze 6 je příliš časově náročná. Na straně osobního počítače, kde implementaci protokolu IP verze 6 zajišťuje .NET Framework je možné budoucí přidání tohoto protokolu jednoduchou změnou kódu.

Požadavky na výslednou aplikaci pro účely diplomové práce byly podrobeny rozbor u možností a analýze realizovatelnosti požadavků. Výsledkem toho je UML diagram tříd, který byl použit jako základ pro další vývoj této aplikace. V tomto diagramu byl kladen požadavek na modulární rozdělení funkcí do samostatných tříd, které s dostatečným oddělením vykonávají svoji činnost. Díky tomuto rozdělení je daleko jednodušší vyměnit nebo přidat nového algoritmu pro šifrování nebo autentizaci. Zároveň lze jednotlivé funkční bloky testovat pomocí takzvaných Unit Testů dostupných v prostředí Visual Studio 2008 Profession Edition, které bylo zvoleno jako vývojové prostředí pro tvorbu této aplikace.

Podrobněji se technikou návrhu a realizace aplikací zabývá zdroj [25], ve kterém se mimo jiné rozebírá způsob testování těchto aplikací, tvorby uživatelské i technické dokumentace a v poslední části se dále zabývá i duševním vlastnictvím a tvorbou licenční politiky.

5.2.1 UML diagram tříd

V příloze jedna je umístěn UML diagram tříd, podle kterého byla vytvořena aplikace pro osobní počítač. Tento diagram byl navržen s ohledem na možné budoucí požadavky vedoucí ke změně v jednotlivých funkčních blocích přenosového kanálu, včetně možnosti úprav nebo přidání dalších šifrovacích nebo autentizačních metod.

Centrální třídou, kterou dědí všechny třídy zabývající se zpracováním informací, je abstraktní třída *BasicDevice.* Součástí této třídy jsou funkce, ve kterých jsou implementovány nejzákladnější operace pro komunikaci funkčních bloků mezi sebou. Tato třída dědí z třídy *RunningDevice* schopnost spustit samostatné vlákno, které bude běžet nezávisle na ostatních vláknech této aplikace.

Mezi třídy, které dědí základní třídu *BasicDevice*, patří třídy *LocalDevice*, *CryptBasicDevice, NetworkBasicDevice* a *Authentication*. Teprve až potomci těchto tříd, kteří mimo jiné implementují rozhraní *IDevice,* jsou plnohodnotnými funkčními bloky, které lze použít v přenosovém kanálu. Třída *LocalDevice* obsahuje metody, které se starají o vytvoření, rušení a práci s virtuálním sériovým portem. K tomu, aby tato třída byla těchto úkolů schopná, disponuje instancí třídy *VSPortAxClass*. Třídy *CryptoDeviceEncryptor* a *CryptoDeviceDecryptor,* jež dědí rodičovskou třídu *CryptBasicDevice*, slouží k šifrování a dešifrování dat. Mimo to je jejich úkolem vytvářet strukturu dat, která se pak rozebere na protější straně. K tomu, aby tyto třídy byly schopny provádět šifrovací činnost, obsahují instanci objektu, která toto šifrování provádí. Aby bylo v budoucnu možno použít různé šifrovací algoritmy, je instance definována jako rozhraní *ICryptoDevices*. Pro zabezpečení detekce správného dešifrování je vytvořena třída *CRC_16_CCITT*, která v sobě implementuje 16bitový cyklický kontrolní součet podle standardu CCITT. Lze použít i další algoritmy, avšak podmínkou je, aby třída, která tento algoritmus implementuje, vycházela z rozhraní *ICRCAlgorithm.* Použité třídy *CryptoDeviceEncryptor* a *CryptoDeviceDecryptor* dědí

z rodičovské třídy definice pracovních streamů, pomocí kterých lze přistupovat k providerům. Dále jmenovaná třída *NetworkBasicDevice* obsahuje definici atributů a služeb, které jsou společnými pro třídy *NetworkClientDevice* a *NetworkServerDevice*. Pro univerzálnost použití jedné z těchto tříd je zavedeno rozhraní *INetworkDevice*, které umožňuje zaměnitelnost obou tříd. Úkolem těchto tříd je vytvořit, udržovat a v případě potřeby zrušit síťové spojení. Poslední třídou, která dědí ze základní třídy *BasicDevice* je třída *AutentikationDevice*, která rovněž obsahuje obecné funkce, které využijí až třídy, které tuto třídu dědí. Příkladem této funkce je *EstablishingKeyAuthentication*, která musí implementovat funkce, které jsou definované v rozhranní *IAuthentication*.

Takzvanou zapouzdřovací třídou je třída *VSPChannel*, jejímž úkolem je soustředit v jedné třídě instance funkčních bloků tak, aby se daly tyto bloky řídit externími příkazy. Těmito externími příkazy jsou volání funkce *StartDevices()* a *StopDevices()*, které spouští a zastavují celý přenosový kanál.

Mimo tyto třídy obsahuje UML diagram další, které jsou například konfigurační, pomocné nebo pro předávání informačních zpráv z funkčních bloků.

5.3 Návrh program pro Rabbit RCM3700

Jak již bylo uvedeno ve čtvrté kapitole, jádrem celého vývojového KITu je 8 bitový mikroprocesor. Z těchto důvodu bylo třeba na tento fakt dbát i při samotné realizaci tak, aby byly všechny aritmetické operace co nejvíce optimalizované pro práci s 8bitovou aritmetikou. Šířka zpracovávaného slova není jediným limitujícím faktorem, mezi další patří například omezené systémové zdroje, jako je velikost operační paměti nebo omezená, více specializovaná instrukční sada, která nedisponuje všemi instrukcemi jako třeba procesor v osobním počítači. Podrobněji se užitím a optimalizací vyššího programovacího jazyka C na mikrokontrolérech zabývá zdroj [26].

Vývojové prostředí, které bylo dodáno spolu s vývojovým KITem, umožňuje pro řídící modul vytvářet programy pouze pomocí jazyka Dynamic C. Tento programovací jazyk vychází z klasického C a není proto problém psát kód právě v tomto jazyku.

Součástí vývojového prostředí jsou také i knihovny. Pro naše účely bylo třeba knihoven pro práci se sériovými porty RS-232, šifrovací operace, tvorbu zabezpečovacích polí zprávy a knihovny pro práci se síťovým rozhraním.

Pro práci se sériovým portem byla potřeba knihovna *RS232.LIB*. Tato knihovna disponuje funkcemi pro práci se všemi dostupnými porty, které naše vývojová deska nabízí.

Jako šifrovací algoritmus byl zvolen AES, který je implementován ve vývojovém prostředí pomocí knihovny *AES_CRYPT.LIB*. Bohužel díky nekompatibilitě mezi starší verzí vývojového prostředí a nejnovějšími verzemi byla zvolena poslední kompatibilní verze, kterou je 1.05.

Práci se síťovým rozhraním umožňuje knihovna *DCRTCP.LIB*, která umí pracovat s čipem, který se stará o připojení do sítě Ethernet. Kromě toho tato knihovna nabízí funkce, které zjednodušují práci s protokoly na odpovídajících vrstvách modelu TCP/IP.

Další potřebnou knihovnou byla knihovna *MATH.LIB*, ve které se nachází funkce *getcrc()* pro výpočet cyklického kontrolního součtu. Tato funkce umožňuje výpočet pouze CRC16 standardu CCITT. Pokud by bylo v budoucnu potřeba použít hashovací funkce MD5, je dostupná v knihovně *MD5.LIB*.

Pro realizaci webového serveru, který by sloužil k nastavení parametrů přenosových kanálů, byla zvolena implementace od společnosti Rabbit, která ho zpřístupňuje pomocí instalace knihovny HTTP.LIB. Součástí této knihovny je implementace skriptovacího jazyka s názvem „ZHTML“, kterým lze vytvářet dynamické stránky. Kvůli již zmiňované kompatibilitě byla v diplomové práci použita verze 1.05.

Pro zajištění bezpečného HTTPS spojení lze webový server rozšířit o protokol SSL, který umožňuje vytvořit šifrované spojení mezi klientem a serverem. Při realizaci diplomové práce byla použita nejnovější možná verze, avšak zároveň kompatibilní, kterou je verze 1.04. Bohužel tato verze má drobné problémy s nejnovějšími webovými prohlížeči.

5.3.1 Vývojový diagram

Vývojový diagram programu pro vývojový KIT je naznačen v příloze dva. Před prací se sockety a sériovou linkou je třeba tyto systémové zdroje nastavit ještě před jejich

použitím. K tomu slouží inicializace. Následně je vybrán jeden z řady přenosových kanálů. Pokud je k socketu tohoto kanálu připojen klient, je testováno, zda je klient autentizován. Pokud ano, tak jsou vykonávány funkce, které zjišťují přijetí dat ze síťového rozhraní nebo sériového portu. V případě, že spojení není autentizováno, jsou vykonávány odpovídající procedury, tak aby autentizace bylo docíleno.

Z diagramu je patrné, že byl navržen s ohledem na vícekanálové přenosy. Díky použitému mikrokontoleru je zřejmé, že vykonávání tohoto kódu bude pouze jednovláknové. Proto se návrh nesnaží vložit do konceptu prvky vícevláknového zpracování, jelikož by musela být zajištěna mezivláknová synchronizace, jejíž zajištění by značně a zbytečně zatížilo mikroprocesor.

V tomto diagramu úmyslně kvůli přehlednosti nejsou znázorněny bloky starající se o autentizaci, synchronizaci času a bloky, které mají za úkol správné fungování webového serveru. Autentizace je realizována v bloku, který se stará o třídění zpráv podle typů. Víceméně autentizace odpovídá obecnému zpracování, jak je naznačeno ve vývojovém diagramu, pouze data nejsou odesílána na odpovídající sériový port, ale jsou zpracovávána v odpovídajících funkcích. Rozhodnutí o tom, která data jsou autentizační, užitečná, nebo servisní, je podle hodnoty v poli „Typ zprávy“ v přijaté zprávě, která má formát podle obrázku číslo 8.

Kromě toho bylo potřeba realizace časovače, který by obstarával odpojení serveru nebo klienta v případě nepovedené autentizace. Tento časovač je založen na hodinách reálného času RTC. Při spojení je uložen čas spojení a průběžně je porovnáván s aktuálním. Pokud by nedošlo do určité doby k úspěšné autentizaci, spojení je ihned ukončeno.

Před inicializací přenosových kanálu a webového serveru lze volitelně spustit synchronizaci lokálního času s několika NTP servery. Pro komunikaci s těmito servery byl použit NTP protokolu, jehož implementace je uvedena v ukázkových kódech, s tím, že některé části kódu byly zjednodušeny, popřípadě pro přehlednost poupraveny.

Stejně jako autentizace a implementace synchronizace s NTP servery i webový server není v diagramu pro jednoduchost znázorněn. Základem jeho fungování je správná inicializace a časté spouštění funkce *http_handler(),* která slouží k pravidelné obsluze příchozích požadavků.

6 Praktická realizace

Po vytvoření odpovídajících diagramů a vybráním vhodných protokolů, algoritmů a vývojových prostředků, včetně těch hardwarových, mohla následovat praktická realizace. Při hardwarové realizaci nebyly použity sériové porty, které lze najít v osobním počítači, protože na daném počítači nebyly. Z těchto důvodů byly použity převodníky USB-RS232 od společnosti WCH s čipem CH341. Tyto převodníky jsou plnohodnotnými sériovými porty podle upraveného standardu RS-232.

Při implementaci bylo třeba dbát na správnou implementaci navržených algoritmů tak, aby nebylo umožněno útočníkovi napadnout přenosový kanál. Zároveň bylo třeba dbát na to, aby nebyly zbytečně spotřebovávány systémové prostředky, jakými jsou výpočetní výkon nebo množství využití operační paměti.

6.1 Realizace aplikace pro Microsoft Windows

Po provedení návrhu, jež je podrobněji popsán v předešlé kapitole, byla provedena realizace aplikace. K realizaci bylo vybráno vývojové prostředí Visual Studio 2008, které vývojářům nabízí stabilní ladící prostředí s možností odchytávání událostí a výjimek. Jako programovací jazyk byl použit jazyk C#, který se v dnešní době hojně využívá v prostředí .NET Framework.

Celá aplikace byla rozdělena na pět částí, z nichž každá část má svůj význam. Ve Visual Studiu k tomuto rozdělení bylo využito možnosti vložit do pracovní složky, která se označuje jako Solution, více projektů, které jsou na sobě prakticky nezávislé. K propojování jednotlivých částí bylo použito takzvaných referencí. To znamená, že výstup jednoho projektu byl vložen dynamicky jako linkovaná knihovna do druhého. Tímto rozdělením došlo k přesnějšímu rozčlenění jednotlivých pracovních etap a k většímu logickému rozdělení celého projektu.

Jádro celé aplikace je uloženo v projektu *VSPChannel.* V tomto projektu jsou definovány všechny hlavní struktury a funkce, které jsou potřeba pro vytvoření, spuštění a zastavení běhu přenosového kanálu. V tomto projektu jsou i funkce sloužící k případné interakci s vyššími vrstvami.

Další projekt s názvem *VSPChannelList* slouží k manipulaci nad objekty typu *VSPChannel*. V této vrstvě jsou definovány metody pro zacházení s objekty typu *VSPChannel* a dále pak funkce pro práci s konfigurací jednotlivých kanálů. Jsou zde definovány i funkce, které načítají nebo ukládají aktuální konfiguraci, popřípadě se starají o předávání aktuální konfigurace objektům typu *VSPChannel*.

Posledním stavebním kamenem funkční aplikace je projekt *ThesisWork*, ve kterém jsou definovány uživatelské funkce a víceméně i vzhled celé aplikace. Tento projekt je jediným, který generuje spustitelný *.exe soubor. Předešlé dva projekty jsou pouze knihovnami a tudíž je nelze spustit. Toto rozložení do projektů je výhodné z pohledu znuvupoužitelnosti kódu. Pokud by vznikl požadavek vytvořit úplně jinou aplikaci se stejně funkčním jádrem, stačilo by v takovém případě použít pouze soubor VSPChannel.dll, jež je výstupem projektu *VSPChannel.* Kolem ní pak naprogramovat například nové uživatelské rozhranní nebo vytvořit projekt typu Windows Service a vytvořit tak systémovou službu jako je například „Zvuk systému Windows“, „Rozpoznávání hardwaru“ a další.

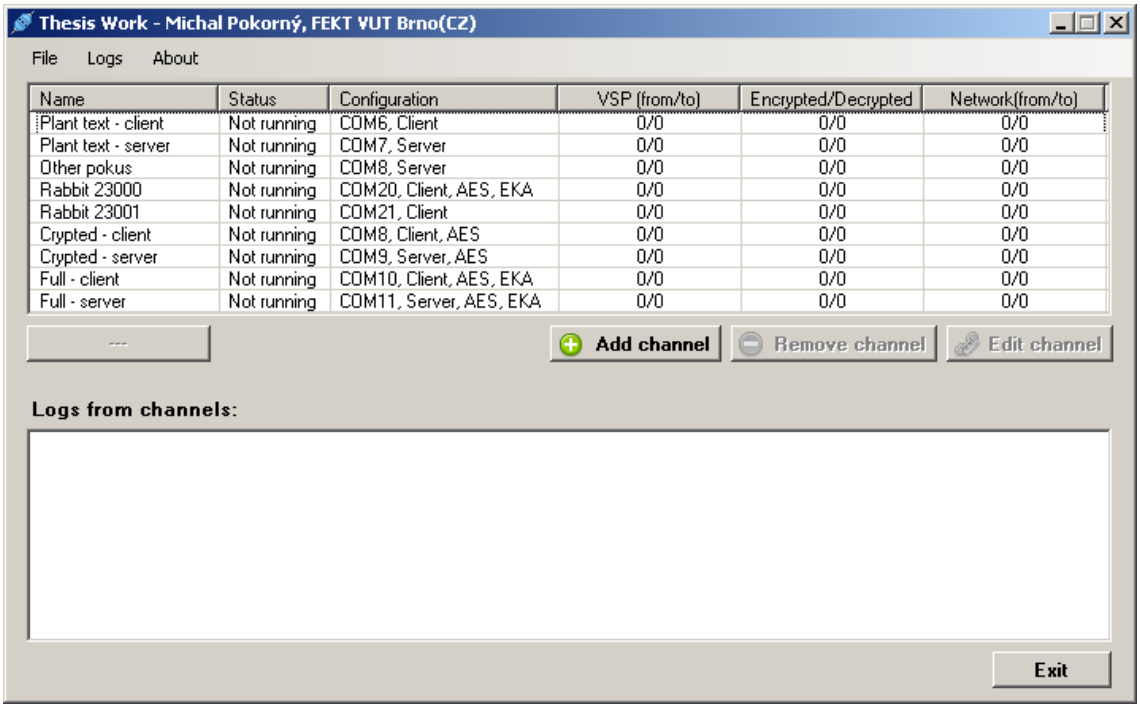
Dalším projektem, který však není pro samotné vytváření nutně nezbytným, ale hodně užitečným, je projekt *TestVSPChannel*. Zde jsou uloženy takzvané unit testy, které slouží k testování správné funkce naprogramovaných funkcí především z projektu *VSPChannel*. V těchto unit testech lze modelovat nejrůznější případy, které mohou nastat při běhu aplikace. Výsledky funkcí v těchto unit testech lze porovnávat s těmi očekávanými. Tento projekt negeneruje žádné dále použitelné výstupní soubory, pouze se užívá k zjištění, zda provedené změny ve funkcích neměly vliv na správné fungování tohoto systému.

Posledním a již pátým projektem je projekt s názvem *Setup.* Jeho úkolem je z výstupů projektů *VSPChannel, VSPChannelList* a *ThesisWork* vzít vygenerovaný výstup a použít jej pro tvorbu instalačního balíčku.

Při návrhu i realizaci byl dbán důraz na zatížení systémových zdrojů, a proto je podstatná část kódu řešena asynchronně nebo spuštěním vlákna, které v případě, že nemá data na zpracování, přejde do stavu nečinnosti, ve kterém zůstává až do okamžiku, kdy ho probudí nějaká událost. Proto bylo například spuštění nebo zastavování přenosového kanálu umístěno do samostatného vlákna pomocí třídy *BackgroundWorker*, tak, aby nebyla snížena nebo znemožněna odezva od uživatelského rozhraní.

6.1.1 Uživatelské rozhranní

Součástí realizace aplikace je i její uživatelské rozhraní, které slouží pro interakci uživatele a přenosových kanálů. Na následujícím obrázku je vidět, z kterých částí se toto rozhraní skládá.



Obrázek 10: Návrh vzhledu aplikace pro osobní počítač

Hlavní ovládací prvky jsou umístěny okolo místa, kde je seznam existujících přenosových kanálů, a ten je umístěn v horní části aplikace. Kromě názvu a aktuálního stavu lze vidět konkrétní nastavení kanálů a aktuální počet přenesených dat v bytech. Při označení zvoleného kanálu jsou nabídnuty možnosti, jako je editace, spuštění, zastavení či smazání kanálu. Podmínkou pro případné smazání je jeho neaktivní stav. Vyvolání editace může být buď zmiňovaným tlačítkem pro editaci, nebo dvojklikem

na zvolený kanál. Pokud je kanál v neaktivním stavu, je zobrazen dialog, ve kterém lze provést odpovídající nastavení. Není-li v neaktivním stavu, je zobrazeno chybové hlášení.

Ve spodní části aplikace je listbox, do kterého jsou vypisovány logovací informace. Součástí těchto informací je na každém řádku čas události, jméno kanálu, ze kterého událost přišla a textová informace o této události. To, zda se budou informace od konkrétních kanálů zobrazovat, lze nastavit v editačním dialogu.

V nejhornější části je pak menu této aplikace. Zde jsou prvky pro ukončení aplikace, uložení logovacích informací a zobrazení informací o aplikaci. V dialogovém okně pro uložení losovacích informací lze zvolit kam a pod jakým názvem budou uloženy. Uložené informace jsou v souboru typu *.xml s odpovídající strukturou.

V rámci editačního dialogu lze nastavit lokální název kanálu, sítové nastavení, nastavení virtuálního sériového portu, šifrovací algoritmus, autentizační algoritmus a nastavení zobrazování logovacích informací. V části pro síťové nastavení lze nastavit, zda se má kanál chovat jako server nebo klient, v případě klienta pak nastavení IP adresy. Dále pak to, který síťový port se má použít a zda se má vyzkoušet dostupnost klienta pomocí ICMP ještě před provedením pokusu o navázání spojení. V části pro editaci virtuálního sériového portu lze nastavit název portu a přenosová rychlost. Nakonec lze zvolit, který šifrovací a autentizační algoritmus má být zvolen, včetně použitého klíče.

6.2 Realizace pro vývojový KIT

Zvolený vývojový KIT má dostatečně obsáhlou dokumentaci [20] a dostatečně rozsáhlý repositář ukázkových kódů, díky kterým bylo pochopení fungování některých algoritmů snazší. Tyto ukázkové kódy jsou již součástí instalace vývojového studia, a proto je nebylo třeba stahovat či instalovat.

Základem celé realizace bylo vytvoření kódu, který má na starosti práci s přenosovými kanály. Z důvodu snazší implementace jsou veškeré události obsluhovány synchronně. V případě asynchronního přístupu by bylo třeba zajistit určitý stupeň synchronizace s ostatními prvky přenosového kanálu, včetně zajištění konzistence stavů ostatních

kanálů. Toto není problém zajistit na vyšších systémech jako je osobním počítač, avšak v 8bitovém mikrokontroléru s omezenou velikostí paměti, výpočetního výkonu a omezenými možnostmi ladění je tato synchronizace složitěji implementovatelná.

Do takto vytvořeného základu byl implementován algoritmus podle obrázku číslo 9. Jedná se o autentizaci, která je identická s tou, jenž je implementována v aplikaci pro osobní počítač. I v této implementaci byl vytvořen časovač, který hlídá spojení, a pokud není spojení do určité doby kladně autentizováno, je spojení okamžitě rozpojeno. Při autentizaci byly použity stavy, které slouží rovněž i jako detekce útoku na autentizační algoritmus. Seznam všech stavů, které byly použity v programu pro modul Rabbit, je v následující tabulce.

Tabulka 3: Stavy kanálu v Rabbit RCM3700

Význam	Konstanta	Hodnota
Spojení nenavázáno	CHANNEL_INIT	0x00
Spojení navázáno	CHANNEL_OPENED	0x01
Začátek autentizace	CHANNEL_AUTHSTART	0x02
Ověření serverem	CHANNEL_RCHECK	0x03
Ověření klientem	CHANNEL_RCHECKED	0x04

Pokud je spojení v posledním stavu, tedy CHANNEL_RCHECKED a je ověření odpovídající strany provedeno korektně, pak je autentizace považována za úspěšnou. V opačném případě je v následujících několika okamžicích po přetečení časovače spojení ukončeno.

Samotná autentizace přímo nevyžaduje nastavení přesného času v hodinách reálného času RTC, avšak pro budoucí užití by bylo vhodné, aby měl tento modul možnost nastavení přesného času. Pro synchronizaci byl zvolen protokol NTP, který se běžně používá v počítačích různého užití. Implementace tohoto protokolu byla převzata z ukázkových kódů a upravena tak, aby odpovídala našim nárokům a požadavkům.

Posledním požadavkem kladeným na toto zařízení, byla možnost vzdáleného nastavení parametrů pomocí webového rozhraní. Pro tyto účely, jak bylo uvedeno výše, byl zvolen modul Rabbit Web. Tento webový server umožňuje vytvářet nezabezpečené

i zabezpečené spojení. K realizaci zabezpečeného spojení byl použit protokol SSL. Ve vývojovém prostředí Dynamic C je implementován jako dodatečný balíček, ve kterém jsou kromě knihoven i programy pro vytváření certifikátů a klíčů.

Pro použití protokolu HTTPS bylo nutno do projektu vložit odpovídající certifikát. Tento certifikát byl vytvořen v programu Certificate.exe, jenž je součástí nainstalovaného balíčku SSL přímo v adresáři instalace vývojového prostředí. Před vlastní tvorbou certifikátu bylo třeba vytvořit certifikát a klíč certifikační autority, na jehož základě pak tento certifikát bylo možno vytvořit.

Po vytvoření těchto odpovídajících prvků, již bylo možno pokračovat při vytváření zabezpečeného webového serveru. Vytvořené stránky mohou být jak statické, tak dynamické. Pro naše účely úpravy a ukládání nastavení je potřeba dynamických webových stránek. Vývojové prostředí pro tyto potřeby nabízí speciální skriptovací jazyk ZHTML. Jeho použití je velmi jednoduché, avšak za cenu nepříliš bohatého množství schopností a někdy až příliš striktních pravidel zápisu.

V příloze číslo tři je možno si prohlédnout výpis konzole vývojového prostředí Rabbit po připojení klienta. Tento výpis nejprve zobrazuje inicializaci sériových rozhraní. Pro testování byly v tomto případě použity sériové porty C a D. Kromě těchto dvou sériových portů disponuje vývojový KIT dalšími sériovými porty, které mají víceméně stejné vlastnosti a chování. Po inicializaci sériových portů následuje inicializace síťového rozhraní. Je vidět, že síťové rozhraní kanálu sériového port C je použito jako serveru, kdežto síťové rozhraní kanálu sériového portu D je použito jako klienta, který se připojuje na vzdálené rozhraní.

Za touto inicializací následuje výpis autentizace, která probíhala ihned po připojení vzdáleného klienta ke kanálu sériového portu C. Při kompilaci zdrojových kódů byl zvolen velmi podrobný výpis ladících informací. Z těchto důvodů jsou ve výpisu i detailní informace, které byly použity v průběhu ladění těchto zdrojových kódů. Těmito detailními informacemi jsou například stavy vyrovnávacích pamětí, včetně jejich obsahů. Dále pak jimi jsou vyjednaný klíč RK, inicializační vektor IV nebo informace týkající se počátků nebo ukončení procesů a stavů.

6.3 Konfigurovatelnost

Od celého systému se očekává, že ho bude možno do určité míry konfigurovat, aniž by bylo třeba provádět znovusestavování binárních spustitelných souborů. Tato konfigurovatelnost byla vyžadována od obou typů komunikačních stran, tj. od aplikace pro osobní počítač i od vývojového KITu.

Nastavení aplikace pro operační systém Windows je možno provádět buď přímo v této aplikaci, nebo pomocí ruční editace odpovídajících konfiguračních souborů, které tuto konfiguraci ukládají v souborech typu xml.

Konfigurovatelnost KITu Rabbit RCM3700 je bohužel menší, než u aplikace pro počítač. Příčin tohoto faktu je rovnou několik. Hlavní příčinou je však to, že jediným programovým prostředkem, který je v modulu Rabbit dostupný, je pouze náš program a pokud se v něm něco má konfigurovat, lze to učinit pouze těmi funkcemi, které jsou v něm naprogramovány. Pokud by tedy měla být zajištěna plná konfigurovatelnost tohoto modulu, značná část naprogramovaného kódu by se musela o tuto schopnost modulu sama postarat.

Z těchto důvodů lze nastavit za běhu jen několik málo vlastností jeho přenosových kanálů. K těm, které lze nastavit patří konfigurace síťového rozhraní a sériového rozhraní. Ostatní parametry lze nastavit při opětovném překladu zdrojových kódů. Mezi tyto needitovatelné parametry patří například nastavení síťového rozhraní, tj. IP adresa, maska sítě, brána a DNS server. Dalšími těmito parametry jsou velikosti vyrovnávacích pamětí, adresy NTP serverů. Popřípadě lze před překladem nastavit, jaká úroveň diagnostických zpráv se má v ladícím okně vypisovat. Úroveň těchto zpráv lze povolit definováním konstant *DEBUG_INFO* a *DEBUG_INFO_DETAIL*.

Jak již bylo uvedeno výše, konfigurace podstatných parametrů je prováděna pomocí webového rozhraní. V příloze číslo čtyři je zobrazeno ukázkové nastavení dvou přenosových kanálů. Toto nastavení bylo rovněž použito v demonstraci připojení v příloze číslo tři.

Před vstupem do tohoto dialogu muselo být zadáno přístupové jméno a heslo, jež je napevno nastaveno ve zdrojových kódech. Ve výsledném zdrojovém kódu je zvoleným uživatelským jménem „admin“ a heslem „top37s1ec5“.

Po provedení požadovaných změn je nové nastavení nejprve webovým serverem zkontrolováno. V případě, že se nevyskytla chyba, je toto nové nastavení uloženo a je proveden reset celého zařízení.

Použitým webovým prohlížečem pro ladění webového rozhraní je prohlížeč Mozilla Firefox Portable Edition[30], který se na rozdíl od klasické edice nemusí instalovat s administrátorskými právy a lze ho přenést na jiný počítač pouhým kopírováním.

6.4 Dokumentace

V rámci realizace bylo vhodné vytvářet odpovídající dokumentaci a vhodně komentovat důležité části zdrojových kódů tak, aby nezasvěcený programátor byl schopen v dostatečně krátkém časovém úseku se v těchto zdrojových kódech zorientovat. Při tvorbě dokumentace pro aplikaci bylo využito schopnosti Visual Studia vytvářet při překladu zdrojových kódů i xml soubor se souhrnem všech nápověd, které byly v zdrojovém kódu napsány. Základní odpovídající struktura těchto nápověd se vygeneruje automaticky po zapsání tří po sobě jdoucích znaků lomítka na odpovídajících místech zdrojového kódu. Těmito místy jsou například záhlaví funkcí nebo atributů tříd.

Vygenerovaný xml soubor byl použít jako vstup speciálních dokumentačních programů. K tvorbě dokumentace pro diplomovou práci byl zvolen program Sandcastle Help File Builder[29]. Výstupem tohoto programu jsou soubory typu *.chm, které se běžně používají pro ukládání nápovědy. Výhodou tohoto programu je velké množství nastavení, díky kterým bylo možno dosáhnout požadované podoby výsledných nápověd. Zároveň bylo možno v nastavení zapnout zobrazování varování, které tvůrce informuje o chybějící nápovědě daných procházených prvků.

6.5 Použité aplikace

Během návrhu a realizace diplomové práce byl použit obsáhlý výčet různorodých programů, aplikací a knihoven. Z důvodu snazšího zorientování byl vytvořen jejich ucelený seznam.

Programy použité pro realizaci:

- Microsoft Visual Studio 2008 - Professional Edition včetně SP1 – vývojové prostředí pro tvorbu aplikace pro operační systém Microsoft Windows
- .NET Framework verze 3.5 včetně SP1 – běhové prostředí, pro které byla aplikace laděna
- Virtual Serial Port ActiveX Control 6.2 – Single Developer Licence – aplikace, která se stará o dynamické vytváření a rušení virtuálních sériových portů
- Hercules Setup Utility 3.1.2 – primárně určen pro jiné účely, ale lze ho využít i pro práci se sériovými porty
- Dynamic C 9.62 včetně modulů AES verze 1.05, SSL verze 1.04 a Rabbit Web verze 1.05 - vývojové prostředí, ve kterém byl vyvíjen řídící program pro Rabbit RCM3700
- Sandcastle Help File Builder – aplikace pro tvorbu dokumentace
- StarUML 5.0.2.1570 – program, který byl použit pro tvorbu UML diagramu

Celý systém byl odlaďován v operačním systému Microsoft Windows XP Professional včetně Servis Packu 3 a nejnovějších aktualizací dostupných v době realizace diplomové práce. Veškerý software byl spouštěn na počítači s jednoprocesorovým jednojádrovým procesorem Intel Pentium Celeron 1.6 GHz a s 2GB operační paměti.

7 Závěr

Výsledkem diplomové práce je kompletní řešení pro bezpečný přenos mezi sériovými porty na vývojovém KITu a virtuálně vytvořenými sériovými porty na osobním počítači.

Základem tohoto řešení je návrh podložený analýzou požadavků. Součástí tohoto návrhu je podrobný popis fungování celého systému, definování jeho funkčních částí a stanovení formátů zpráv, které se přenáší mezi jednotlivými účastníky komunikace. Pro potřebu tvorby zdrojových kódů byly vytvořeny diagramy, podle kterých se dále postupovalo. UML diagram tříd byl vytvořen pro účely zrealizování aplikace pro osobní počítač s operačním systémem Microsoft Windows. Pro účely tvorby řídícího programu vývojového prostředí Rabbit RCM3700 byl vytvořen vývojový diagram popisující fungování přenosových kanálů. Dále byly v návrhu zvoleny vhodné šifrovací a autentizační algoritmy, jejichž úkolem bylo vytvoření bezpečného přenosového kanálu.

Byla rovněž vytvořena aplikace, která je schopna vytvořit virtuální sériové porty pomocí komponenty od společnosti Eltima, a data z těchto portu bezpečně přenést na vzdálený sériový port. Tato aplikace byla dále dolaďována tak, aby byla schopna vytvářet trvalá a bezpečná spojení mezi ní a vývojovým KITem Rabbit RCM3700.

V diplomové práci byl rovněž realizován řídící program pro již zmíněný vývojový KIT. Realizace byla provedena na základě předešlého návrhu. Při této realizaci však byly zjištěny nové poznatky, díky kterým musel být původní návrh v určitých detailech poopraven.

V rámci diplomové práce byly řešeny i různé problémy a úskalí. Tyto problémy byly především technické povahy, i když některé z nich měly i netechnický charakter, jako například licenční podmínky užívání programů a knihoven, použitých v rámci předešlé realizace aplikace i řídícího programu. Při řešení těchto problémů byla zjištěna celá řada užitečných poznatků, které byly zhodnoceny při realizaci dalších částí tohoto přenosového systému.

Výstupem této práce je navržený a realizovaný systém, který lze na základě této práce dále rozvíjet. Bohužel použitý modul RCM3700 je na samé hranici použitelnosti

a pro větší rozšíření funkcionality by již nemusel vyhovovat. Podle dokumentace je maximální velikost binárních dat v paměti omezen na velikost přibližně 200kB, která byla při realizaci překročena. Z těchto důvodů musela být část dat přesunuta i do pomalejší sériové paměti. Zároveň byly odděleny data a program, které jsou nyní uloženy zvlášť. Tato separace však měla za následek drobné snížení odezvy webového serveru.

V případě většího nasazení lze pro řídící modul navrhnout jiný rozšiřující modul, než je použitá vývojová deska a patřičně upravit zdrojový kód pro tento nový rozšiřující modul.

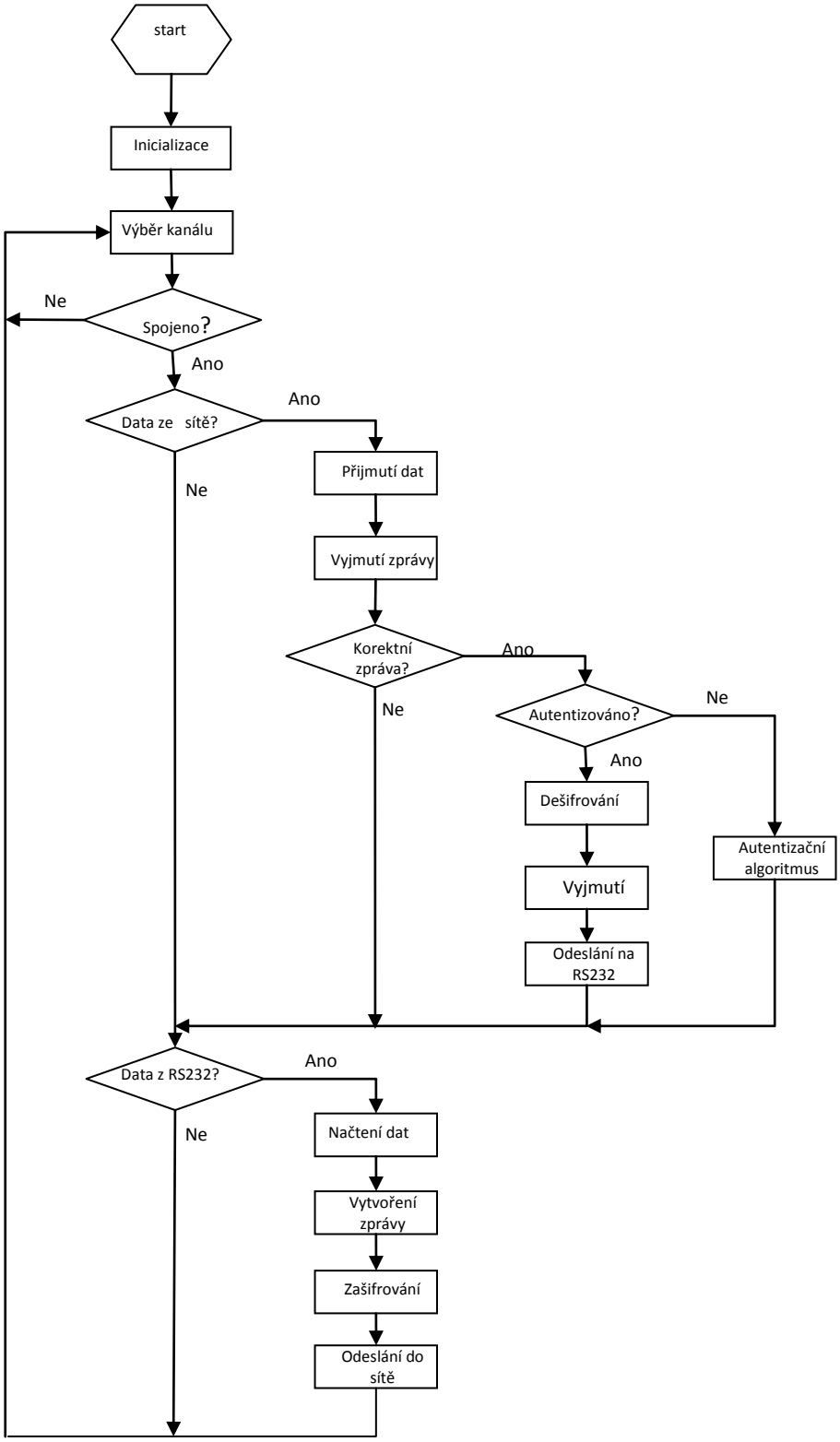
Diplomová práce vedla k seznámení se s analýzou zadání, tvorbou návrhů, praktickou realizací a s vyhledáváním a řešením vzniklých problémů. Zároveň došlo k seznámení s použitými protokoly, standardy a aplikacemi pro tvorbu a ladění zdrojových kódů s odpovídající kvalitou.

Přílohy

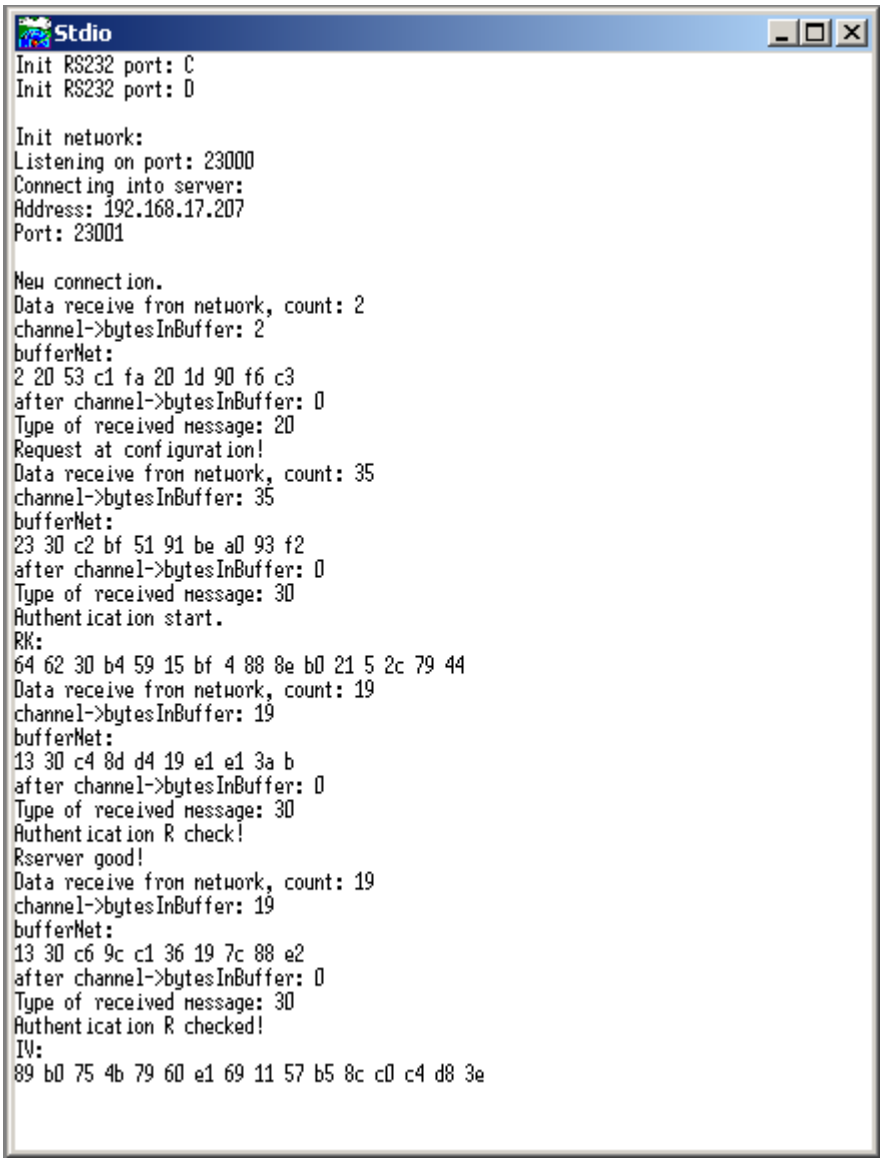
Příloha 1: UML diagram

Příloha je vložena v deskách nebo je dostupná elektronicky na CD.

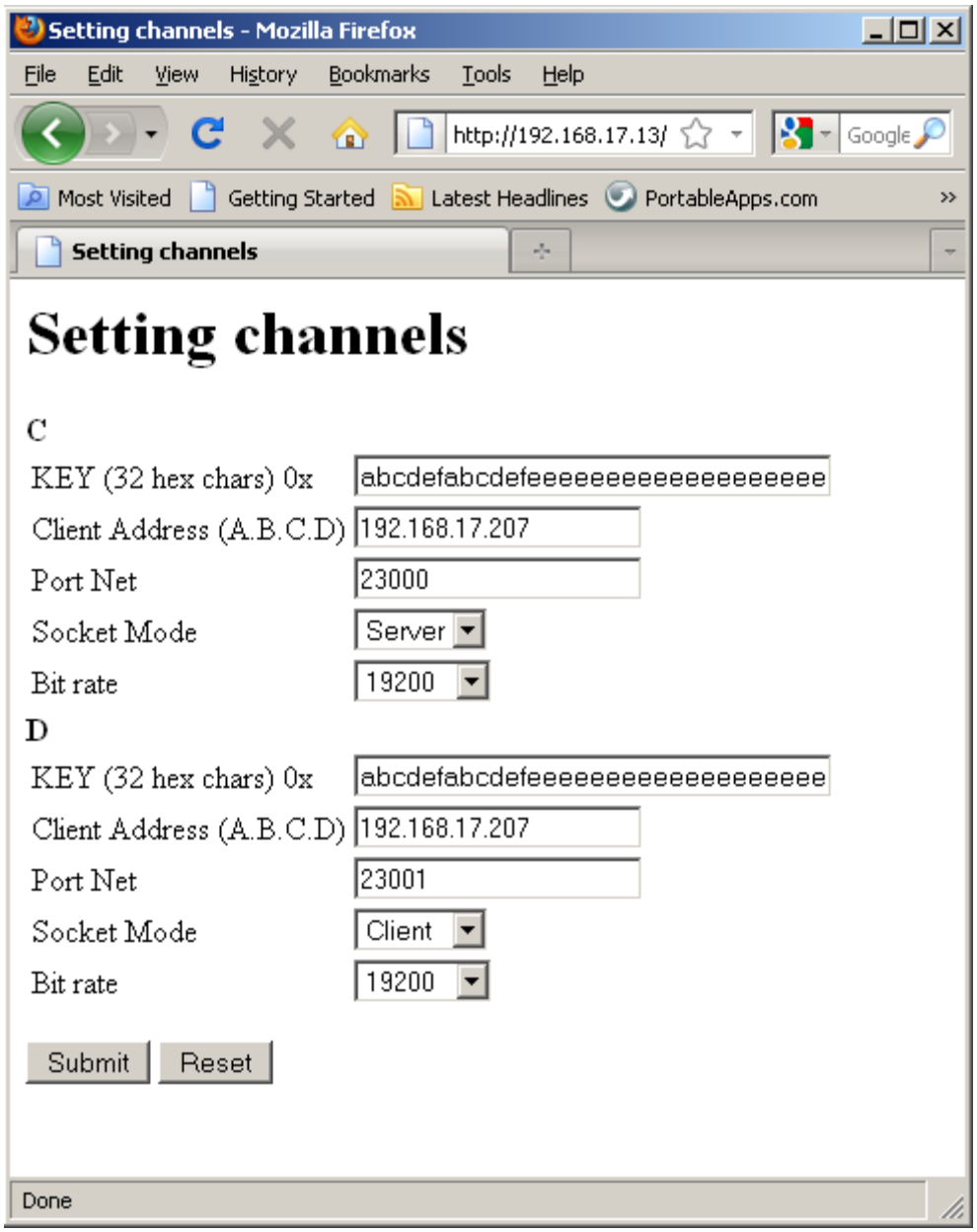
Příloha 2: Vývojový diagram programu pro Rabbit RCM3700



Příloha 3: Výpis konzole vývojového prostředí



Příloha 4: Konfigurace Rabbit RCM3700



Použitá literatura

[1] Mueller, Scott. *Osobní počítač. Hardware, Upgrade, Opravy*. 2003. 862s. ISBN 80-7226-796-5

[2] Marek, Rudolf. *Učíme se programovat v jazyce Assembler pro PC*. 2003. 232s. ISBN 80-722-6843-0

[3] Filka, M. *Optické sítě*. Skripta VUT FEKT. Brno,VUT FEKT UTKO 2007.

[4] Kainka, Burkhard. *Využití rozhranní PC*. 1999. 133s. ISBN 80-902059-3-3

[5] Matoušek, David. Udělejte si z PC – generátor, čítač, převodník, programátor. 2001. 175s. ISBN 80-7300-036-9

[6] Matoušek, David. Udělejte si z PC – užitečný stroj a ovládejte porty ve Windows. 2002. 223s. ISBN 80-7300-072-5

[7] HW server - RS232 [online]. 1997-2009 [cit. 2009-12-06]. Dostupný z WWW: <http://hw.cz/rs-232>.

[8] Straka, Pavel. *Internetový protokol IPv6*. 2008. 359s. ISBN 978-80-904248-0-7

[9] Osterloh, Heather. *TCP/IP – Kompletní průvodce*. 2003. 512s. ISBN 80-86297-34-8

[10] LIČEV, Lačezar, MORKES, David. *Procesory - architektura, funkce, použití*. 1999. 260 s. ISBN 80-7226-172-X

[11] BARRETT, Daniel, SILVERMAN, Richard. *SSH – Kompletní průvodce*. 2003. 556 s. ISBN 80-7226-852-X

[12] KOUTNÝ, Martin, HOŠEK, Jiří. *Návrh kryptografického zabezpečení systémů hromadného sběru dat* [online]. c2007 [cit. 2009-12-11]. Dostupný z WWW: <http://www.elektrorevue.cz/cz/clanky/informacni-techologie/0/navrh-kryptografickeho-zabezpeceni-systemu-hromadneho-sberu-dat/>.

[13] Garfinkel, Simpson. *PGP: Pretty Good Prvacy*. 1998. 373 s. ISBN 80-7226-054-5

[14] PIRKL, Josef. Síťové programování pod Windows a programování Internetu. 2002. 360s. ISBN 80-7232-145-5

[15] .NET Framework Overview [online]. 2009 [cit. 2009-12-12]. Dostupný z WWW: <http://www.microsoft.com/net/overview.aspx>.

[16] .NET Framework Developer Center [online]. 2009 [cit. 2009-12-12]. Dostupný z WWW: <http://msdn.microsoft.com/cs-cz/netframework/default%28en-us%29.aspx>.

[17] MSDN [online]. 2009 [cit. 2009-12-08]. Dostupný z WWW: <http://msdn.microsoft.com/>.

[18] ARLOW, Jim, NEUSTADT, Ila. *UML a unifikovaný proces vývoje aplikací*. 2008. 567s. ISBN 978-80-251-1503-9

[19] Rabbit - Products [online]. 2009 [cit. 2009-12-08]. Dostupný z WWW: <http://rabbit.com/products/>.

[20] The RCM3700 RabbitCore Ethernet microprocessor core module [online]. 2009 [cit. 2009-12-08]. Dostupný z WWW: <http://rabbit.com/documentation/docs/manuals/RCM3700/index.htm>.

[21] Multi-Port Serial-to-Ethernet [online]. 2009 [cit. 2009-12-10]. Dostupný z WWW: <http://rabbit.com/products/MultiS2E_Kit/>.

[22] Sběr dat z elektroměrů prostřednictvím Internetu [online]. 2009 [cit. 2009-12-10]. Dostupný z WWW: <http://pandatron.cz/?1092&sber_dat_z_elektromeru_prostrednictvim_internetu>.

[23] KAINKA, Burkhard. *USB – Měření, řízení a regulace pomocí sběrnice USB*. 2003. 247s. ISBN 80-7300-073-3

[24] Virtual Com Port ActiveX Control [online]. 2009 [cit. 2009-12-09]. Dostupný z WWW: <http://www.eltima.com/products/vspax/>.

[25] GUNDERLOY, Mike. Z kodéra vývojář – Nástroje a techniky pro opravdové programátory. 2007. 287s. ISBN 978-80-251-1517-6.

[25] MANN, Burkhard. *C pro mikrokontroléry*. 2003. 280s. ISBN 80-7300-077-6.

[26] JAMSA, Kris. *Programování na WEBU*. 1996. 661s. ISBN 80-86097-03-X.

[27] World Wide Web Consortium (W3C) [online]. 2010 [cit. 2010-05-10]. Dostupný z WWW: <http://www.w3.org/>.

[28] The Network Time Protocol [online]. 2009 [cit. 2010-05-10]. Dostupný z WWW: <http://www.ntp.org/>.

[29] Sandcastle Help File Builder [online]. 2009 [cit. 2010-05-12]. Dostupný z WWW: <http://shfb.codeplex.com/>.

[30] Mozilla Firefox, Portable Edition [online]. 2010 [cit. 2010-05-18]. Dostupný z WWW: <http://portableapps.com/apps/internet/firefox_portable>.