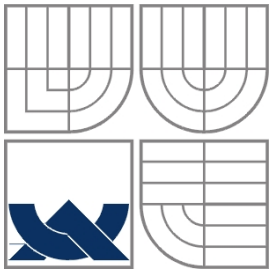
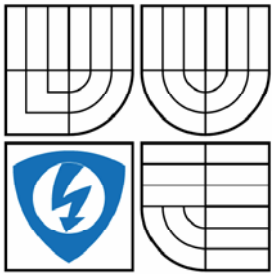


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

BEZDRÁTOVÝ PŘENOS OBRAZU MEZI PC

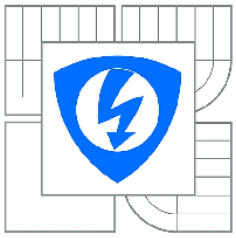
WIRELESS TRANSFERT OF IMAGE BETWEEN THE PCS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
ŠTEFAN MIŠÍK

VEDOUCÍ PRÁCE
SUPERVISO
Ing. LUDĚK ČERVINKA

BRNO 2012



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Štefan Mišík	***ID:***	125270
Ročník:	3	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Bezdrátový přenos obrazu mezi PC

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je vytvořit aplikaci, která bude mít za úkol navázat bezdrátové spojení mezi dvěma či více počítači. Tato aplikace bude na jedné straně mít připojenou kameru, která bude snímat obraz a následně jej bude odesílat dalším stanicím, kde bude zobrazen. V případě nutnosti může být obraz vhodně zkomprimován.
Dále bude úkolem studenta provést měření rychlosti přenosu dat a porovnat jej s teoretickými hodnotami, zjistit maximální dosah, na který možno odesílat data a provést statistku úspěšnosti odeslaných dat.

Dále bude úkolem studenta se zabývat rychlostí přenosu při různých typech přenosu obrazu.

DOPORUČENÁ LITERATURA:

ŠONKA, M., HLAVÁČ, V., BOYLE, R.: Image Processing, Analysis, and Machine Vision. Thomson, Toronto, 2008. 829 p. ISBN 978-0-495-08252-1.

Termín zadání:	6.2.2012	***Termín odevzdání:***	28.5.2012

Vedoucí práce: Ing. Luděk Červinka
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Dokument se zabývá možným řešením problematiky bezdrátového přenosu obrazu mezi počítači a jeho realizací jako aplikace pro operační systém *Microsoft Windows*.

Klíčová slova

Bezdrátová komunikace, přenos obrazu, Programování pro *Microsoft Windows*

Abstract

The thesis discusses the possible solution to the problem of wireless video transmission between computers and its implementation as an application for *Microsoft Windows*.

Keywords

Wireless communications, Image transfer, *Microsoft Windows* programming

Bibliografická citace:

MIŠÍK, Š. *Bezdrátový přenos obrazu mezi PC*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 53s. Vedoucí bakalářské práce byl Ing. Luděk Červinka.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma *Bezdrátový přenos obrazu mezi PC* jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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 části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **28. května 2012** ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Luďkovi Červinkovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **28. května 2012**

……
podpis autora

Obsah

1 ÚVOD

1.1 Vytýčené ciele

Cieľom práce je zvoliť vhodný prostriedok bezdrôtového prenosu dát a vytvoriť aplikáciu schopnú prenášať obraz medzi počítačmi, pomocou zvoleného prenosového prostriedku. Návrh aplikácie zahŕňa vytvorenie komunikačného jadra a aj grafického rozhrania.

Kritériami pri voľbe prenosového prostriedku bude okrem jeho schopnosti naplniť požiadavky vyžadované pre prenos obrazu aj jeho dostupnosť a cena, čo je samozrejmé a bežné kritérium riešenia pri realizácii akéhokoľvek projektu.

1.2 Úvod do problému

Systém pre prenos obrazu nájde v súčasnosti mnoho uplatnení, nie len v priemysle, ale aj v bežnom živote. Dnes už je možné nájsť zabudovanú kameru takmer na každom novšom notebooku a dostupnosť externých zariadení na snímanie obrazu je veľmi dobrá a ich ceny sú prijateľné. Ak sa k tejto skutočnosti pridá možnosť prenosu bezdrôtovými prenosovými cestami spektrum praktických aplikácii, v ktorých je použiteľný, sa môže len rozšíriť.

Ak uvážime len použitie v priemysle, je možné vďaka takémuto systému monitorovať procesy alebo priestory. Pripojením ďalších systémov, v rátane systémov umožňujúcich počítačové spracovávanie obrazu a počítačové videnie, by sa naskytla možnosť automatizácie procesov.

Realizácii systému musí predchádzať proces návrhu a najmä voľby bezdrôtového prostriedku schopného prenášať dáta s dostatočnými parametrami prenosu, nato aby bol prenos obrazu realizovateľný. Na riešenie tejto problematiky budú zamerané ďalšie kapitoly.

2 ANALÝZA PROBLÉMU

V tejto kapitole budú teoreticky rozobrané jednotlivé súčasti zadania projektu a analyzované možné postupy riešenia.

2.1 Formulácia problému

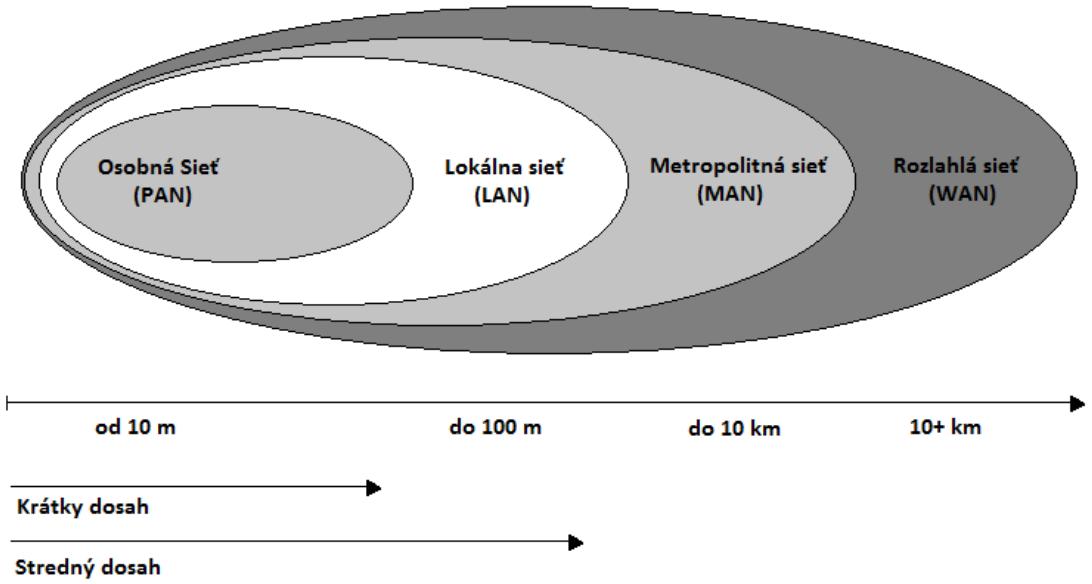
Prvým krokom k realizovaniu bezdrôtového prenosu obrazu je uskutočnenie bezdrôtového prenosu všeobecných dát. K tomuto účelu je v súčasnosti možné využiť široké spektrum prostriedkov. Výber správneho prostriedku bezdrôtového prenosu je kľúčovým pre výsledné parametre prenášaného obrazu. Najväčšiu rolu samozrejme zohráva prenosová rýchlosť, ktorá ovplyvňuje kvalitu, veľkosť obrazu a plynulosť (v zmysle počtu prenesených snímkou za sekundu). Nezanedbateľnú rolu zohrávajú však aj iné parametre prenosu. Veľkosť časového oneskorenia a straty údajov pri prenose môže tiež zásadne ovplyvniť výsledný obraz. Preto je nutné zvážiť, najmä na základe vedomosti o spôsobe implementácie tvoreného riešenia do praxe, či sú prípustné rôzne nedokonalosti v prenášanom obraze a na základe tejto znalosti sprísniť, či zmierniť požiadavky kladené na prenosový prostriedok.

Keďže je možné predpokladať, že na prenos kompletných obrazových informácii by bolo nutné zabezpečiť veľmi veľkú prenosovú rýchlosť, je nevyhnutné počítať s nutnosťou kompresie prenášaných dát. Existuje však mnoho rôznych kodérov – dekodérov, ktoré plnia tento účel a prirodzene niektoré splňujú požiadavky kladené na kvalitu prenášaného obrazu a množstva dát prenášaných prenosovým prostriedkom rôzne. Kvôli tomu je nutné porovnať niektoré možnosti a z nich vybrať optimálne riešenie.

Súčasťou riešenia má byť aj počítačová aplikácia zabezpečujúca prenos medzi počítačmi. Pred začatím procesu tvorenia je nutné zvoliť parametre výslednej aplikácie. Základom je stanoviť platformu, na ktorej má aplikácia byť spustiteľná. Nasleduje výber prostriedkov určených na tvorenie aplikácii, čo zahŕňa voľbu programovacieho jazyka, pracovného prostredia a vývojových prostriedkov (použitý *framework*, knižnice...).

2.2 Prostriedky bezdrôtového prenosu dát

Bezdrôtový prenos dát je možné realizovať rôznymi prenosovými prostriedkami, ktoré sú popísané v [4] a [5]. Tieto prostriedky môžeme rozdeliť do skupín podľa ich dosahu ich signálu (znázornené na Obr. 2.1).



Obr. 2.1 Dosah bezdrôtových sietí

Medzi rozľahlé siete s dosahom nad 10 km patria satelitné komunikačné siete, metropolitné siete zahŕňajú mobilné siete a siete založené na štandarde IEEE 802.16, akou je *WiMAX*. Lokálne siete môžu byť postavené na štandarde 802.11, čo sú siete nazývané *Wi-Fi*. Osobné siete sú zastúpené štandardami *Bluetooth* a *ZigBee*.

Keďže jednou z požiadaviek na realizáciu projektu je aj nízka cena a dosah siete do 100 m bude dostačujúci, nemá zmysel uvažovať nad bezdrôtovými sieťami typu MAN a WAN, ktoré všetky pracujú v licencovaných frekvenčných pásmach.

V nasledujúcich kapitolách budú rozobraté len bezdrôtové siete s fyzickou vrstvou založenou na štandardoch IEEE 802.11 a IEEE 802.15, ktoré umožňujú prevádzku v bezlicenčných pásmach.

2.2.1 Bezdrôtová lokálna sieť 802.11 (Wi-Fi)

Bezdrôtové stanice komunikujúce prostredníctvom tejto siete môžu komunikovať v dvoch režimoch:

Nezávislá konfigurácia (ad hoc) – sieť nemá žiadnu podpornú infraštruktúru ani prístupový bod, je tvorená len komunikujúcimi stanicami, ktoré komunikujú medzi sebou priamo.

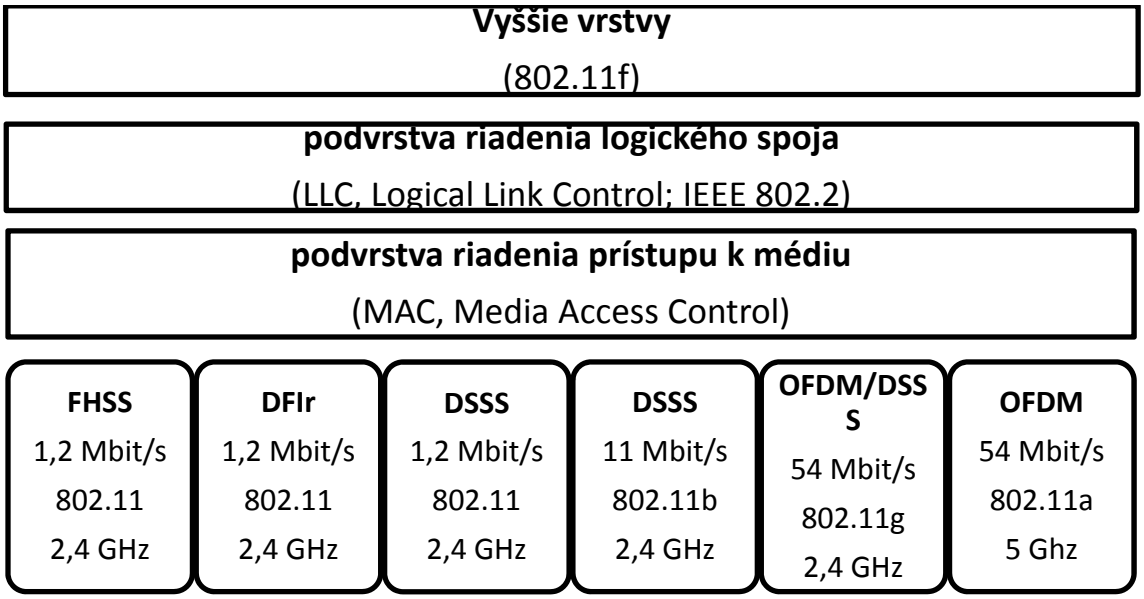
Konfigurácia s distribučným systémom (DS) – Jednotlivé stanice komunikujú s prístupovým bodom (AP – *Access Point*). Stanice v dosahu jedného AP spolu s týmto prístupovým bodom vytvárajú celok zvaný súbor základných služieb (BSS – *Basic*

Service Set). Jednotlivé súbory základných služieb môžu byť prepojené distribučným systémom a vytvárajú súbor rozšírených služieb (ESS – *Extended Service Set*). Prístupový bod v tejto sieti predstavuje bezdrôtové premostenie komunikujúcich staníc a distribučnej siete. Ak je v dosahu stanice viac prístupových bodov, stanica si vyberie jeden a s ním udržuje spojenie. Spôsob výberu prístupového bodu stanicou a presunu medzi stanicami je definovaný v norme.

Distribučný systém predstavuje druh chrbtovej siete, ktorá, ako je popísané v norme, môže byť realizovaná aj bezdrôtovo.

Okrem spomínaných dvoch režimoch komunikácie klientov je ešte možné vytvoriť aj tzv. *Mesh* sieť, v ktorej bezdrôtové siete nemusia komunikovať len s prístupovým bodom, ale aj priamo medzi sebou. Prístupové body v tejto sieti vystupujú ako smerovače paketov a hľadajú najkratšiu cestu k príjemcovi. Tento spôsob komunikácie je popísaný v norme IEEE 802.11s.

Na Obr. 2.2 je znázornené usporiadanie vrstiev siete postavenej na štandarde IEEE 802.11, kde prostredné dve vrstvy predstavujú spojovú vrstvu v ISO OSI modely a najnižšie vrstvy predstavujú rôzne fyzické vrstvy ISO OSI modelu, ktoré definujú rôzne časti normy IEEE 802.11.



Obr. 2.2 Spojová a fyzická vrstva WLAN

Ako z obrázka vidno teoretické rýchlosti prenosu dát v tejto sieti dosahujú až 54 Mbit/s. V súčasnosti sú už aj novšie, nie však príliš rozšírené (a v niektorých prípadoch ešte nedokončené) normy IEEE 802.11n a IEEE 802.11ac, ktoré umožňujú niekoľko násobne vyššie prenosové rýchlosti. V prípade prvej je to do 300 Mbit/s a v druhom prípade niekoľko Gbit/s. Oba tieto štandardy na dosiahnutie tak vysokých prenosových rýchlostí využívajú jeden z princípov takzvaných chytrých antén – MIMO (*Multiple-InputMultiple-Output*). Pri ktorom sa pridávaním ďalších antén na vysielač a/alebo na príjmač zvyšuje priepustnosť siete a aj jej dosah. Dôvodom urýchlenia prenosu pri

použití MIMO je oproti štandardnému jedno - anténovému systému nielen potlačenie ale využitie odrazených vĺn. Tento proces je však náročný a vyžaduje spracovanie signálu v digitálnych signálových procesoroch. V bežných systémoch s jednou anténou na príjmači aj vysielači spôsobujú odrazené vlny rušenie a v konečnom dôsledku útlm signálu (anglický výraz pre tento jav je *multipath distortion*).

2.2.2 Bezdrôtové siete postavené na štandarde IEEE 802.15

Týmto názvom je možné súhrnne označiť hneď niekoľko rôznych bezdrôtových sietí s krátkym dosahom signálu na úrovni osobných sietí, teda do 10 m. V nasledujúcich podkapitolách budú rozobrané niektoré z významných zástupcov rodiny IEEE 802.15.

2.2.2.1 Bluetooth (IEEE 802.15.1)

Je to bezdrôtová sieť pracujúca v bezlicenčnom pásme 2,4 GHz, ktorá vo svojej prvej verzii dosahuje reálne rýchlosti prenosu dát do 720 kbit/s. Špecifikácia *Bluetooth* taktiež obsahuje podporu rôznych služieb (ako prenos hlasu, súborov, kontaktných informácii, dát...) prostredníctvom zabudovanej podpory kvality služieb.

Technológia bezdrôtovej siete *Bluetooth* využíva pásmo 2,4 GHz metódou rozprestretého spektra s preskakovaním medzi kanálmi. Toto preskakovanie sa deje asi 1600 krát za sekundu a je semi - náhodné (náhodná postupnosť je odvodená od fyzickej adresy zariadenia).

Topológia sietí založených na štandarde *Bluetooth* pozostáva v základnej podobe (*pikonet*) z jedného prvku typu *master* a jedného až siedmych prvkov typu *slave*. Obmedzenie je spôsobené skutočnosťou že prvok typu *master* môže naraz komunikovať maximálne so siedmymi prvkami typu *slave*. Ďalšie rozšírenie siete je možné pridaním prvku *master*/*slave*, ktorý tým, že môže byť pre jedno zariadenie *slave* a pre iné zariadenia *master*, umožňuje prepojiť niekoľko *pikonetov* do celku zvaného *scatternet*.

Štandard *Bluetooth* existuje aj v špeciálnej forme s nižšou spotrebou pre zariadenia napájané z batérie. V tejto modifikácii má však výrazne nižší dosah. Používa sa v mobilných zariadeniach.

V súčasnosti dosahujú bežne používané siete postavené na štandarde *Bluetooth* maximálne prenosové rýchlosti 3Mbit/s a dosah signálu je okolo 10 m, v niektorých špeciálnych prípadoch energicky náročných modifikácii do 100 m. Na vývoji nových verzií sa stále pracuje a do budúcnosti sa počíta s rýchlosťami 480 Mbit/s až 1 Gbit/s.

2.2.2.2 UWB a IEEE 802.11.3a

Skratka v názve označuje *Ultra Wide Band* (šírka kanálu až 500 MHz), čo umožňuje dátové prenosy na veľmi krátke vzdialenosti (do 10 m) s rýchlosťou do 480 Mbit/s.
Tento štandard aj keď má zaujímavé charakteristiky sa však poriadne neujal, čo je spôsobené najmä skutočnosťou, že normalizácia nebola dokončená, kvôli dlhodobým nezhodám v normalizačnej skupine.

2.2.2.3 ZigBee (IEEE 802.11.4)

Jedná sa o špecifikáciu bezdrôtových sietí zameranú na malú spotrebu energie a spoľahlivosť prenosu. Z toho vyplývajú malé prenosové rýchlosti (maximálne 250 kbit/s – podľa [8]). Štandard je zameraný na použitie v priemysle najmä na bezdrôtovú komunikáciu so snímačmi, ale uplatňuje sa aj v domácej automatizácii.

2.2.3 Porovnanie jednotlivých bezdrôtových technológii

Na záver je v Tab. 2.1 súhrnné porovnanie spomínaných bezdrôtových technológii. Pre porovnanie je tu spomenutý aj infračervený prenos *IrDA*, ktorého použitie je však komplikované, keďže vyžaduje viditeľnosť prijímača aj vysielača a aj ich správne nasmerovanie.

Tab. 2.1 Porovnanie jednotlivých bezdrôtových technológii

Typ	Frekvencia (GHz)	Kapacita na fyzickej vrstve	Dosah
802.11b (Wi-Fi)	2,4 - 2,485	11 Mbit/s	100 m
802.11a	5,1 – 5,3 5,725 – 5,825	54 Mbit/s	50 m
802.11g	2,4 – 2,485	54 Mbit/s	80 m
802.15.1 (*Bluetooth*)	2,4	3 Mbit/s	10 m
802.15.4 (*ZigBee*)	2,4 868/915 MHz	20/40/250 kbit/s	20 m
IrDA	Ir: 850 nm	4 Mbit/s	10 m

2.3 Referenčný model ISO OSI

Jedná sa o model architektúry komunikačných protokolov počítačových sietí. Keďže súčasťou zadania projektu je aj navrhnutie počítačovej aplikácie, ktorá bude využívať služby počítačových sietí, je nutné spomenúť aj túto tému.

Model pre riadenie sieťovej komunikácie zložený z viacerých vrstiev sa ujal už v počiatkoch počítačových sietí. Referenčný model ISO OSI sa skladá zo siedmych vrstiev (jednotlivé vrstvy sú znázornené na Obr. 2.3, prevzaté z http://en.wikipedia.org/wiki/File:Osi-model-jb.png, originál vytvoril JB Hewitt (Johnblade) na en.wikipedia), pričom každá vrstva, okrem najvyššej, poskytuje služby nadradenej vrstve, využívajúc služby poskytované nižšou vrstvou (okrem najnižšej vrstvy). Každá vrstva u jedného účastníka komunikácie komunikuje s vrstvou na tej istej úrovni u druhého účastníka komunikácie. Táto komunikácia prebieha na základe

predpísaných zoznamov pravidiel - protokolov. Jednotlivé vrstvy referenčného modelu ISO OSI sa môžu skladať aj z podvrstiev, čo je vidno aj na Obr. 2.2, kde spojovú vrstvu z ISO OSI modelu predstavujú podvrstvy: riadenie logického spoja a riadenie prístupu k médiu.

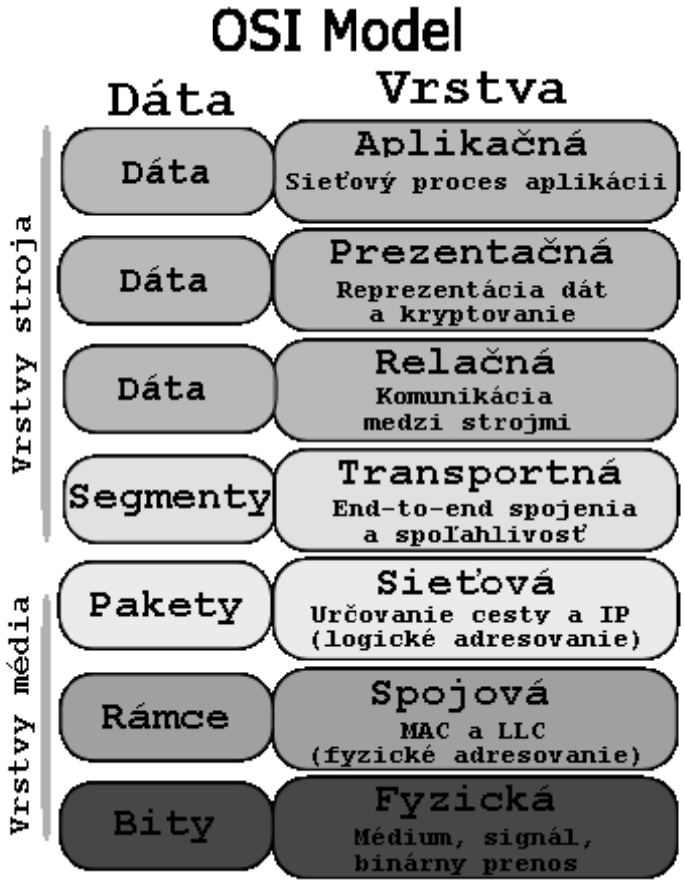
Funkcia každej z vrstiev bude stručne rozobraná, na základe popisu v [4], v nasledujúcej podkapitole.

2.3.1 Vrstvy OSI modelu

Vrstvy OSI modelu sa rozdeľujú do dvoch základných kategórii:

Lokálne postupy (vrstva 1-3) – Nižšie vrstvy sa starajú o komunikáciu sieťovú infraštruktúru. Na Obr. 2.3 sú označené alternatívnym názvom vrstvy média.

Koncové postupy (vrstva 4-7) – Vyššie vrstvy umožňujú väzbu komunikačných aplikácii a sietí. Môžu byť tiež nazývané vrstvy stroja.



Obr. 2.3 Vrstvy OSI modelu

Aplikačná vrstva (*application layer*) predstavuje rozhranie s užívateľom, medzi služby poskytované touto vrstvou patrí prenos správ, identifikácia komunikujúcich partnerov, zistenie pripravenosti partnera, dohoda o mechanizmoch zabezpečenia a kontroly, synchronizácia a výber syntaxi komunikačného protokolu.

Funkciu tejto vrstvy môže vykonávať aj človek.

Prezentačná vrstva (*presentation layer*) má za úlohu transformovať prichádzajúce dáta do jednotnej podoby vhodnej pre aplikačnú vrstvu, to sú napríklad dátové štruktúry. Nezaoberá sa významom správ, len ich transformuje.

Relačná vrstva (*session layer*) udržuje relácie spojení medzi komunikujúcimi partnermi, jedna prezentačná vrstva môže byť pripojená na viac relačných vrstiev. Relačná vrstva poskytuje služby vytvorenia a ukončenia relačného spojenia, riadenie interakcie (jednosmerná, obojsmerná striedavá a obojsmerná súčasná), synchronizáciu relačného spojenia, oznamovanie výnimočných správ.

Transportná vrstva (*transport layer*) je postavená medzi aplikáciu (užívateľa) a sieť, takže aplikácia sa nemusí starať o usporiadanie a prenosové prostriedky siete.
Transportná vrstva poskytuje:

transportnú službu bez spojenia – len prenos blokov (protokol UDP)

transportnú službu so spojením – naviazanie, udržovanie a záver virtuálnych okruhov (protokol TCP)

Transportná služba so spojením zabezpečuje, že dáta poskytované vyšším vrstvám sú neporušené, v správnom poradí a nie sú duplicitné. Keďže výstupom je spojitý prúd dát, nie je možné určiť v akých blokoch dáta posielal odosielateľ, ak je nutné mať aj túto informáciu, je nutné túto informáciu do komunikačného protokolu začleniť, napríklad vopred dohodnutou dĺžkou blokov, ukončovacími znakmi alebo vložením informácie o dĺžke jednotlivých blokoch.

Transportná vrstva taktiež umožňuje identifikáciu aplikácii bežiacich na komunikujúcich zariadeniach pomocou čísla portu.

Sieťová vrstva (*network layer*) umožňuje adresovanie a smerovanie dátových jednotiek – paketov (datagramov) v komplexnej sieti. Bežne sa tu využíva IP protokol, ktorý na identifikáciu jednotiek v sieti využíva IP adresy, nazývané tiež sieťové alebo logické adresy.

Spojová vrstva (*data link layer*), jednou z jej funkcii je fyzické adresovanie, adresuje dátové rámce jednotkám v rámci jednej siete. Taktiež spolupracuje pri riadení prístupu k fyzickému médiu prenášajúcemu dáta.

Fyzická vrstva (*physical layer*) zabezpečuje fyzickú komunikáciu medzi komunikujúcimi jednotkami. Definuje fyzickú interpretáciu bitov na symboly.

2.4 Záver k OSI modelu a bezdrôtovým sieťam

Takmer všetky spomínané prostriedky bezdrôtového prenosu dát úzko súvisia s OSI modelom, keďže štandardy z rodiny IEEE 802.x definujú práve len najnižšie dve vrstvy modelu OSI (fyzickú a spojovú) a vyššie vrstvy sú pre všetky štandardy spoločné.

Zo spomínaných sú výnimkami len siete *Bluetooth* a *ZigBee*, ktoré definujú aj vyššie vrstvy referenčného modelu. A však aj tu platí, že samotná aplikácia o tejto skutočnosti nemusí mať žiadne informácie a môže pracovať rovnakým spôsobom ako na iných sieťach.

2.5 Kompresia obrazu

Ako je spomenuté v [9] pri multimediálnej komunikácií by bol bez nejakého opatrenia objem prenášaných dát priveľký, pre realizáciu komunikácie v reálnom čase. Preto sa správy odosielané komunikačným kanálom musia podrobiť kompresii.

2.5.1 Dátová reprezentácia obrazových informácii

Aby bolo možné obraz poslať z jedného miesta na iné je nutné tento obraz najskôr previesť do vhodnej podoby. Keďže počítačová technika, ktorá má byť pri prenose použitá, je schopná pracovať len s číslami, je nutné obrazové informácie popísať týmto spôsobom. Takto spracované obrazové dáta je možné buď priamo posielať prenosovými prostriedkami, alebo prípadne ešte pred poslaním vhodne skomprimovať.

Aj tu existuje mnoho spôsobov ako popísať obrazové informácie číslami. V technickej praxi sa najčastejšie využíva spôsob popisu farebného obrazu, kde každý bod v obraze je popísaný tromi hodnotami: intenzity červenej, zelenej a modrej zložky (RGB). Tento spôsob popisu obrazu je síce výhodný pre spracovanie obrazu, ale nie je príliš vhodný na jeho prenos, keďže časť informácii je pre ľudský zrak nadbytočná.

Ďalším možným, často používaným, spôsobom prevodu obrazu sú formáty YUV a YC_bC_r. Matematický popis prevodu medzi formátom YUV a formátom RGB je vyjadrený sústavou rovníc (2.1). Formáty YUV a YC_bC_r sú v princípe veľmi podobné a líšia sa len mierne odlišnými koeficientmi pre výpočet chrominančných zložiek (U, V a C_b, C_r) a tiež v skutočnosti, že formát YUV primárne popisuje spôsob prenosu obrazovej informácie analógovými signálmi, pričom formát YC_bC_r určuje číselnú interpretáciu obrazových dát. Formát YUV, aj keď sa týka analógovej techniky a nie počítačovej, je tu uvádzaný z dôvodu, že tento názov sa zo zvyklosti používa aj v počítačovej technike.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}169 & -0{,}331 & 0{,}5 \\ 0{,}5 & -0{,}419 & -0{,}081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad \textbf{(2.1)}$$

Zložka Y v tomto formáte má fyzikálny význam v podobe intenzity jasu bodu vnímaného okom. Matematicky je vypočítaný zo zložiek RGB ako Y = 0,299R + 0,587G + 0,114B. Zložky U a V majú význam rozdielových zložiek medzi modrou zložkou RGB a Y zložkou vynásobenou koeficientom 0,565 (U zložka) a medzi červenou zložkou RGB formátu a Y zložkou vynásobenou konštantou 0,713 (V zložka).

Obrazové dáta vo formáte YUV sú vhodnejšie pre kompresiu obrazu, keďže koeficienty popisujúce jeden obrazový bod pri použití tohto formátu majú väčší význam v kontexte spôsobu akým ľudské oko vníma obraz. Je zistené, že ľudské oko je citlivejšie na zmeny zložky Y ako na zmeny zložiek U a V. Táto skutočnosť sa využívala už v analógovej televízii, kde v systéme PAL bola zložka Y prenášaná vo frekvenčnom pásme 5 MHz a zložky U a V boli prenášané v pásmach širokých len 1,3

MHz. Tým už v analógovej technike bola dosiahnutá akási kompresia prenášaných informácií.

2.5.2 Spôsoby kompresie obrazu

Keď už je obraz prevedený do jeho číselnej reprezentácie je možné na tieto dáta aplikovať špeciálne postupy za účelom zníženia redundancie prípadne aj irelevancie obrazových dát.

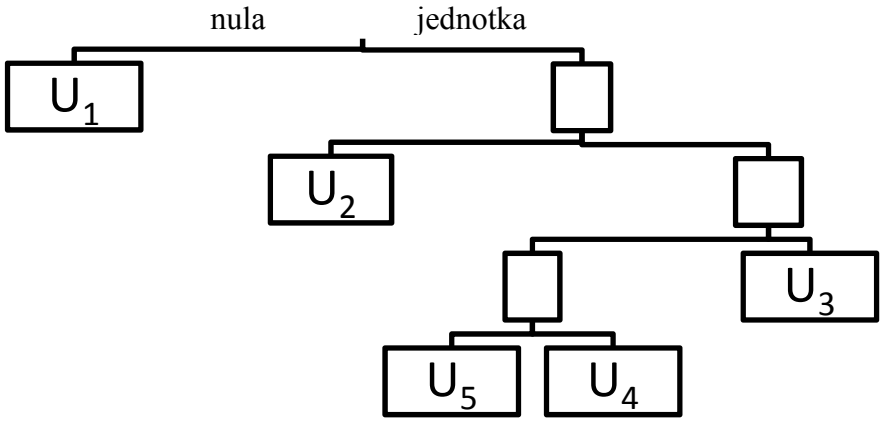
Pod pojmom zníženie redundancie sa väčšinou myslí bezstratová kompresia, to znamená, že zdrojový kód po zakódovaní takouto kompresiou je možné obnoviť do identickej podoby, v akej bol pred zakódovaním.

Prvým a najstarším spôsobom bezstratovej kompresie je Morseova abeceda. To je vďaka skutočnosti, že najkratšie sekvencie sú pridelené najčastejšie sa vyskytujúcim písmenám (v anglickom jazyku).

Najčastejšie využívané bezstratové metódy kompresie využívajú metódy prideľovania nerovnomerne dlhých kódových symbolov ku zdrojovým symbolom na základe pravdepodobnosti výskytu jednotlivých zdrojových symbolov. Takéto spôsoby kompresie sa nazývajú *nerovnomerné kódy*.

Jedným z často využívaných spôsobov bezstratovej kompresie pomocou nerovnomerných kódov je *Huffmanov kód*. Vďaka skutočnosti, že vykazuje zároveň aj vlastnosti prefixového kódu je tento kód jednoznačne dekódovateľný.

Kódovanie sa nazýva prefixové kódovanie, ak žiadne kódové slovo nie je prefixom iného slova. Túto skutočnosť na prvý pohľad vidno na znázornení binárneho vyhľadávacieho stromu (Obr. 2.4): ak vždy jedna vetva vedie ku vyhľadávanému symbolu zdrojového kódu a druhá vetva toho istého rozvetvenia vedie ku ďalšiemu rozvetveniu (okrem najspodnejšej vetvy) jedná sa o prefixový kód.



Obr. 2.4 Binárny vyhľadávací strom pre Huffmanov kód

Vyššie spomenutý príklad Morseovej abecedy nie je prefixovým kódom, keďže na jednoznačné dekódovanie je používaný tretí symbol – dlhšia časová medzera – ktorá oddeľuje symboly zdrojového kódu.

Na dosiahnutie vyšších kompresných pomerov bez straty podstatných informácii sa používa stratová kompresia. Stratovú kompresiu je na rozdiel od bezstratovej možné použiť len na dáta, o ktorých vieme aký druh informácie prezentujú. Ak je toto známe, je možné rozhodnúť, ktoré časti zdrojového kódu je možné odstrániť bez straty podstatnej informácie. K tomu slúžia rôzne fyziologické modely, ktoré boli vytvorené skúmaním spôsobu, akým človek vníma podnety z okolia. Tieto modely teda určia, ktoré informácie nie sú pre zmysly človeka podstatné a teda ich absenciu človek takmer nepostrehne.

V nasledujúcich podkapitolách budú spomenuté niektoré zvažované stratové kompresie obrazu.

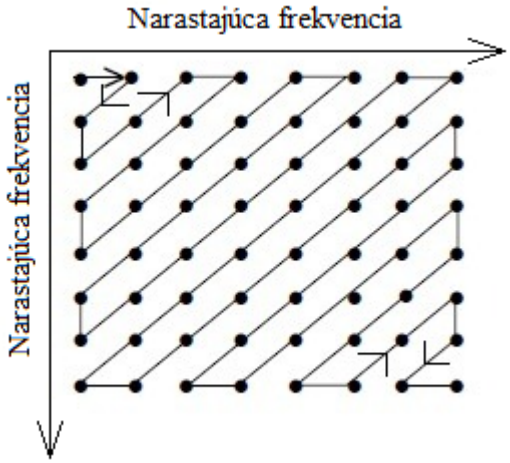
2.5.2.1 Odporučenie JPEG

Odporučenie JPEG (*Joint Photography Expert Group*) je určené pre kompresiu statických obrazov. Tento spôsob kompresie je vhodný pre akýkoľvek kompozitný formát obrazových dát, napríklad pre formát YUV ale aj pre formát založený na primárnych farebných zložkách RGB.

Každá farebná zložka je rozdelená do segmentov 8 x 8 bodov a následne sú tieto segmenty transformované pomocou diskrétnej kosínusovej transformácie (DCT). Kosínusová transformácia použitá pri JPEG kompresii je popísaná vzťahom (2.2).

$$F(k,l) = \frac{1}{2^{N-1}} \sum_{m=-N+1}^{N-1} \sum_{n=-N+1}^{N-1} f(m,n) \cdot cos\left[2\pi \frac{m \cdot k}{2N-1}\right] \cdot cos\left[2\pi \frac{n \cdot l}{2N-1}\right], \quad \textbf{(2.2)}$$

kde N je dĺžka hrany štvorca, ktorý tvorí obrazový segment, v tomto prípade je to 8 obrazových bodov. Diskrétna funkcia f(m,n) vytvorená z obrazového segmentu tak, že segment je skopírovaný symetricky podľa osí x a y s prekrytím obrazových bodov ležiacich na osách. Diskrétna funkcia F(k,l) je potom diskrétna kosínusová transformácia funkcie f(m,n). Hodnoty koeficientov s narastajúcou frekvenciou sú odčítané spôsobom, ktorý je naznačený na Obr. 2.5. Táto postupnosť je potom podrobená vyhodnoteniu dĺžok postupností rovnakých hodnôt RLE (*Run Length Encoding*) okrem prvého koeficientu, ktorý má fyzikálny význam ako jednosmerná zložka osvetlenia v segmente. Ten má mnohokrát väčšiu hodnotu ako ostatné koeficienty. Jeho hodnota je vyjadrená ako hodnota rozdielu od koeficientu ležiaceho na rovnakom mieste v predošlom segmente. Tým sa ušetrí miesto potrebné na uchovanie tejto hodnoty, keďže sa predpokladá, že na snímkach reálneho sveta sa nachádza omnoho menej ostrých hrán ako postupných prechodov.



Obr. 2.5 Postupnosť odčítania koeficientov

Výsledné koeficienty sa ešte komprimujú bezstratovým *Huffmanovým kódovaním*. Veľkou výhodou formátu JPEG je najmä jeho relatívna jednoduchosť a skutočnosť, že proces dekódovania je identický, len v obrátenom poradí, ako proces kódovania.

2.5.2.2 Odporučenie MPEG

Odporučenie MPEG (*Motion Picture Experts Group*) je kompresia, ktorá je používaná pre záznam pohyblivých obrazov na pamäťové médium (MPEG 1 je určený pre CD-ROM, rýchlosť jeho bitového toku nepresahuje 1,5 Mb/s).

Obrazové dáta vstupujúce do kodéru musia byť vo formáte YUV, keďže na zvýšenie kompresného pomeru sa využíva aj detekcia pohyblivých objektov a táto detekcia je vykonávaná práve na zložke Y, chrominančné zložky U a V sú tu zanedbávané.

Kompresia MPEG dosahuje viac ako dvojnásobnú efektivitu kompresie odstránením redundancie nielen vo vnútri snímku ale aj medzi jednotlivými snímkami. Obraz je rozdelený na pohyblivé a statické oblasti a na obe časti je aplikovaný podobný komprimačný postup ako pri kompresii JPEG.

Pre dosiahnutie vysokého stupňa kompresie sa predpokladá, že postupnosť po sebe nasledujúcich obrazových rámcov vykazuje značnú podobnosť. Tejto podobnosti sa využíva pri striedaní rámcov typov I, P a B, ktorých význam je nasledujúci:

rámec I (*Initial*) je kódovaný nezávisle, spôsob jeho kompresie je takmer identický s kompresiou JPEG, dosahovaná kompresia tohto rámcu je asi 7:1,

rámec P (*Predicted*) vyjadruje zmenu obrazu oproti jednému predchádzajúcemu obrazovému rámcu, kompresný pomer môže nadobudnúť hodnotu až 20:1,

rámec B (*Bidirectionally Interpolated*) vyjadruje vzťah k jednému predchádzajúcemu snímku a jednému nasledujúcemu snímku typu I alebo P. Tento rámec dosahuje najvyšší kompresný pomer, a to až 50:1.

Ďalšie zvýšenie kompresného pomeru kódovania časovej postupnosti rámcov sa dosiahne použitím tzv. *makro blokov*. Makro blok je časť obrazu, ktorej obsah sa časom nemení, ale posúva sa po obraze ako celok. Makro blok môže identifikovať pohyblivý

objekt alebo jeho časť. Rámec typu P definuje pre každý makro blok jeden vektor pohybu. Pri rámci typu B sú pre každý makro blok definované dva vektory pohybu.

Výpočet vektoru pohybu je jedným z naj náročnejších úloh pri kompresií videa do formátu MPEG. Na výpočet sa väčšinou používa len jasová zložka Y a chrominančné zložky U a V sa zanedbávajú.

Je zrejmé že existuje mnoho rôznych implementácií kodéru a dekodéru formátu MPEG. Samotné odporučenie MPEG sa zameriava len na stanovenie základných pravidiel a nie je prekážkou pri inovácií funkčných vlastností, ktoré sú závislé na implementácií kodéru a dekodéru.

3 VOĽBA METÓD RIEŠENIA PROBLÉMU

Každý proces voľby medzi určitými možnosťami vedie v reálnom svete ku kompromisu, vzhľadom na skutočnosť, že väčšinou nie je možné dosiahnuť stav, aby všetky parametre výsledku boli maximálne dobré. Je možné sa k tomuto stavu priblížiť, ale väčšinou je cena takejto možnosti privysoká (cena nemusí byť len v zmysle finančnej ceny). Preto je nutné s ohľadom cieľové uplatnenie výsledku práce zvoliť optimálny kompromis z ponúkaných možností.

3.1 Voľba spôsobu prenosu dát

Z vybraných prostriedkov bezdrôtového prenosu dát popísaných v kapitole 2.2 je nutné zvoliť cenovo dostupný prostriedok, ktorého parametre budú postačovať požiadavkám na prenos obrazu. Keďže je tu aj požiadavka na čo najnižšiu cenu môžu byť z výsledného výberu vynechané všetky prenosové prostriedky pracujúce v licencovaných frekvenčných pásmach. Tým zostávajú všetky siete postavené na štandardoch IEEE 802.11 a IEEE 802.15. Aby bolo možné rozhodnúť správne, ktorú technológiu použiť, je nutné najskôr odhadnúť požiadavky kladené na prenosovú cestu, pri prenášaní obrazu touto cestou.

3.1.1 Parametre prenášaného obrazu

Obraz vo forme video záznamu má nasledujúce parametre ovplyvňujúce požiadavky kladené na prenosový prostriedok:

Bitová hĺbka – určuje počet bitov definujúcich jeden obrazový bod, určuje farebnú rozlíšiteľnosť obrazu

Rozlíšenie obrazu – počet bodov v jednom obrazovom riadku a počet riadkov v obraze.

Počet snímkou za sekundu

Kompresia

Prenos obrazu zahrňuje nielen prenos pohyblivého videa, ale aj prenos stacionárnych snímkou. Ak chceme usudzovať o požiadavkách kladených na prenosový prostriedok, nie je možné tento parameter ani v tomto prípade vynechať. V tomto prípade nahradíme tretí bod obrátenou hodnotou doby, za ktorú chceme mať obrázok prenesený.

Pre účely prvého priblíženia požadovaných parametrov nebude uvažovaná kompresia (aj keď tá môže požiadavky kladené na prenos obrazu zásadne ovplyvniť) a tiež nebudú uvažované rôzne obslužné dáta, ktoré je tiež nutné preniesť spolu so „surovými“ (*raw*) obrazovými dátami.

Pre výpočet priepustnosti prenosového média potrebnej na prenos surových obrazových dát sa vynásobia prvé tri parametre obrazu spomínané na začiatku tejto kapitoly, ako vyjadruje vzťah (3.1).

$$min.priepustnosť = BH * VR * HR * PSzS \quad \left[\frac{bit}{s}\right]. \qquad (3.1)$$

BH - bitová hĺbka [bit]
VR - vertikálne rozlíšenie [-]
HR - horizontálne rozlíšenie [-]
PSzS - počet snímkou za sekundu [s-1]

Pre rôzne využitia prenášaného obrazu sú však kladené rôzne požiadavky na parametre obrazu. Preto budú v nasledujúcich podkapitolách analyzované niektoré možné využitia systému prenosu obrazu.

3.1.1.1 Použitie pri sledovaní priestorov (bezpečnostná kamera)

Táto aplikácie nie je príliš náročná na požiadavky kladené na parametre prenášaného obrazu. Bežne je tu postačujúca bitová hĺbka obrazu 8 bitov, rozlíšenie 640 x 480 a počet snímkou sa pohybuje do 15 snímkou za sekundu.

Z toho vyplýva požadovaná prenosová rýchlosť: 8*640*480*15 = 36,864Mbit/s.

3.1.1.2 Použitie v priemysle

Pri použití systému na prenos obrazu v priemysle sa môžu na prenášaný obraz klásť rôzne požiadavky v závislosti na aplikácii systému. Ak má systém zachytávať a prenášať informácie o rýchlom procese, pričom je dôležitá spojitosť informácii (nepostačujú len stacionárne snímky rýchleho procesu v určitý stanovený čas), sú požiadavky na rýchlosť prenosového média o to vyššie. Na druhej strane môže nastať prípad kedy stačí aj jedna snímka každých niekoľko desiatok sekúnd.

3.1.1.3 Použitie v bežnom živote

Tu sa vyžaduje najmä oku príjemný obraz, čo zahŕňa dostatočné rozlíšenie (aspoň 768 x 576), bitovú hĺbku 24 bitov (plne farebný obraz) a najmä pre plynulosť obrazu rýchlosť snímkovania nad 25 snímkou za sekundu. Z týchto parametrov vychádza bitová rýchlosť potrebná na prenos takéhoto obrazu približne 265,42 Mbit/s.

3.1.1.4 Prenos statických snímkou

Ak uvážime prenos statických snímkou, pričom bude postačujúci čas na prenos snímku okolo 5 sekúnd, vyjde prenosová rýchlosť potrebná na prenos plnofarebnej snímky s rozlíšením 640x480 približne 1,5 Mbit/s.

3.1.2 Nutnosť kompresie obrazu

Vidno, že s prenosom surových obrazových dát by, aj pri miernejších požiadavkách na parametre obrazu, mali aj najrýchlejšie uvažované technológie problém. Preto je nevyhnutné použiť kompresiu obrazu.

V súčasnosti poskytujú stratové kompresné formáty možnosť zmenšiť objem prenášaných dát oproti surovým obrazovým dátam až 100 násobne, v závislosti na nastavenej kvalite obrazu, keďže sa jedná o stratové formáty, klesá spolu s objemom prenášaných dát aj kvalita obrazu. Tu zase treba nájsť vhodný kompromis medzi objemom prenášaných dát a kvalitou obrazu.

3.1.3 Záver k voľbe prenosového prostriedku

Ak použijeme kompresiu obrazu, ktorá bude poskytovať zmenšenie objemu prenášaných dát v priemere 50 krát, je možné prenášať aj oku príjemný obraz rýchlosťou okolo 5 Mbit/s. Požiadavky však väčšinou nebudú nastavené na takúto vysokú kvalitu obrazu, preto môžeme podľa Tab. 2.1 usúdiť, že požiadavkám budú vyhovovať takmer všetky verzie *Wi-Fi* sietí a aj *Bluetooth* siete.

Siete založené na štandarde IEEE 802.15.3, aj keď by tiež vyhovovali požiadavkám, sú vyhovujúce, keďže technické prostriedky nutné na realizáciu týchto sietí sú málo rozšírené.

Na druhej strane sú siete založené na štandardoch *Wi-Fi* a *Bluetooth* výhodné aj pre dobrú dostupnosť a relatívne nízku cenu technických prostriedkov nutných k realizácií týchto sietí.

Keďže v zadaní je projektu je uvedené že prenos dát má byť realizovateľný aj medzi viac ako dvomi počítačmi, je nutné zvoliť technológiu *Wi-Fi*, ktorá umožňuje dostatočnú rýchlosť prenosu aj pre viac ako jedno spojenie prenášajúce obrazové dáta súčasne.

3.2 Výber vhodnej metódy kompresie obrazu

Ako bolo spomenuté v predošlej kapitole, z dôvodu veľkých objemov prenášaných dát pri prenose nekomprimovaného obrazu, sa kompresia zdá byť nutnou súčasťou bezdrôtového prenosu obrazu.

Z kompresných formátov uvažovaných v kapitole 2.5 je nutné zvoliť optimálnu voľbu. Pričom je zrejmé, že nie je nutné vybrať tú s najlepším kompresným pomerom, keďže prenosový kanál zvolený na realizáciu bezdrôtového prenosu je schopný dostatočne rýchlo prenášať dáta.

Ak by sa bral do úvahy len kompresný pomer rozhodne by zvíťazil formát MPEG, ktorý je prispôsobený na kompresiu videa. Avšak táto voľba prináša aj mnoho nevýhod. Jednou z najväčších sa zdá byť nutnosť väčších vyrovnávacích pamätí a s tým súvisiace väčšie dopravné oneskorenie. Toto je z toho dôvodu, že pri kompresií podľa odporučenia MPEG vznikajú tzv. snímky typu B, teda, ako je spomínané aj v kapitole 2.5.2.2, snímky obojsmerne interpolované. Z toho vyplýva že pri príchode takejto

snímky je nutné počkať na príchod najbližšej nasledujúcej snímky typu I alebo P, pričom všetky ostatné dáta je nutné uchovať vo vyrovnávacej pamäti. Ďalším negatívom tohto formátu je skutočnosť, že neskomprimované obrazové dáta je nutné pred kompresiou formátom MPEG previesť do formátu YUV. Za predpokladu, že dáta posielané z kamery budú vo formáte RGB je tento prevod ďalším výpočtovým zaťažením počítača. Pri zvážení skutočnosti, že samotná kompresia MPEG je výpočtovo náročná, by sa mohlo dôjsť k záveru, že takáto kompresia by mohla počítač zaťažiť do takej miery, že by nebol možný uskutočňovať túto kompresiu v reálnom čase.

Preto bola ako najvhodnejšia zvolená kompresia JPEG, ktorej kompresný pomer by mal postačovať na prenos relatívne plynulého videa. Aj v literatúre ([9]) sa spomína, že tento druh kompresie sa v niektorých konferenčných aplikáciách úspešne využíva práve pre jeho jednoduchosť a relatívne dobrú účinnosť. Ďalšou výhodou je skutočnosť, že je schopný komprimovať akýkoľvek kompozitný formát obrazových dát, či už sa jedná o RGB, YUV alebo YC_bC_r ako aj čiernobiely obraz. A to práve z toho dôvodu, že každý kanál komprimuje zvlášť a rovnakým spôsobom.

3.3 Voľba postupu tvorenia aplikácie

Prvým a najzákladnejším parametrom vytvorenej aplikácie je určite určenie na akej platforme a operačnom systéme má byť spustiteľná. Keďže zo zadania vyplýva, že aplikácia má byť spustiteľná na osobnom počítači, ponúka sa široké spektrum operačných systémov.

Kvôli všeobecnej popularite a rozšírenosti bol zvolený operačný systém *Microsoft Windows*. Vývojové prostredie *Visual Studio* určené na vývoj aplikácii pre tento operačný systém ponúka mnoho spôsobov ako vytvárať aplikácie, pomocou rôznych programovacích jazykov.

Keďže existuje viacero možností ako postupovať pri vytváraní aplikácie pre operačný systém *Microsoft Windows*, budú nasledujúce podkapitoly venované rozboru zvažovaných možností.

3.3.1 Microsoft Visual C#

Špecifikácie tohto jazyku sú popísané v [7] okrem toho boli použité aj iné zdroje: [2] a [3].

Je to čisto objektovo orientovaný programovací jazyk založený na syntaxe jazykov C a C++, z ktorých preberá aj niektoré kľúčové slová a podobné sú aj metodiky programovania.

Tento jazyk poskytuje pohodlný, moderný a bezpečný spôsob vytvárania aplikácii. Od jazykov ako C a C++ sa líši najmä tým, že programy vytvorené pomocou tohto jazyku sú spúšťané na tzv. *.NET Framework* – u, čo je v súčasnej dobe už integrálna súčasť operačného systému *Windows*. Tento *framework* sa skladá z *Common language runtime* (CLR) a súboru unifikovaných tried. CLR je implementácia medzinárodného štandardu *Common language infrastructure* (CLI) vytvorená firmou Microsoft.

Zdrojový kód jazyku C# (na rozdiel od jazykov ako C a C++) je podľa štandardu CLI prekladaný do kódu nazvaného *Intermediate language* (IL).

Ak je požadované vykonanie tohto kódu, musí byť najskôr načítaný do *Common language runtime* (CLR), ktorý rozhodne o ďalších operáciách. Ak sú všetky požiadavky splnené, je *Intermediate language* (IL) preložený pomocou *Just-in-time* (JIT) prekladača prevedený do vlastného strojového kódu počítača.

Common language runtime poskytuje aj mnoho ďalších služieb. Jednou z nich je *Automatic garbage collection*, čo je proces, pri ktorom sú z pamäte uvoľnené všetky objekty bez referencií v kóde. Vďaka tomu nie je nutné sa starať o správne uvoľnenie všetkej alokovanej pamäte. Nevýhodou pri takomto spôsobe správy pamäte je skutočnosť, že nie je možné presne určiť kedy bude pamäť uvoľnená, prípadne kedy bude zavolaný deštruktor objektu. Nie príliš príjemnou skutočnosťou je aj fakt, že deštruktor bude vykonaný iným vláknom (vláknom, na ktorom beží *garbage collector*) ako to ktoré, objekt vytvorilo. Môže dokonca nastať aj situácia, kedy za celý beh programu nebude zavolaný deštruktor objektu, ktorý už nemá žiadnu referenciu. Výhodou prostredia *.NET Framework* je jeho pokročilá správa výnimiek (*exception*), ktorá umožňuje relatívne jednoduché odchytávanie a spracovávanie chýb vzniknutých počas behu programu.

3.3.2 Microsoft Visual C/C++

Zdrojový kód napísaný pomocou týchto jazykov sa na rozdiel od C# väčšinou prekladá priamo do strojového kódu. Toto môže byť výhoda ak je požadovaná rýchlosť a efektivita vykonávania kódu, ale vytváranie komplexnej aplikácie je tu viac komplikované.

3.3.3 Zvolený postup tvorby aplikácie

Aj keď programovací jazyk *Microsoft Visual C#* poskytuje pohodlný a moderný spôsob tvorby aplikácii, je možné, že skutočnosť, že výsledný kód aplikácie by bol spúšťaný pomocou *Common language runtime* (CLR), tzv. manažovaný kód, by mala nepriaznivý dopad na výkon výslednej aplikácie.

Najmä kvôli tejto skutočnosti bol nakoniec zvolený programovací jazyk *C*. Toto síce prináša do tvorby väčšiu komplikovanosť postupu, výhody však zrejme prevyšujú nad nevýhodami.

4 REALIZÁCIA RIEŠENIA

Pred samotnou realizáciou je nutné naplánovať vytváranie jednotlivých častí aplikácie a teoreticky analyzovať princípy funkcie vytváranej aplikácie aby proces vytvárania nebol chaotický ale organizovaný a výsledok mal konkrétnu myšlienku.

4.1 Analýza aplikácie

Podľa zadania má byť aplikácia realizovaná v tejto časti práce schopná komunikovať prostredníctvom počítačovej siete s inou aplikáciou spustenou na inom počítači pripojenom k sieti.

K tomuto účelu bude aplikácia s výhodou využívať nástroje zvané *sokety*, ktoré zjednocujú komunikáciu po rôznych sieťach (ale aj len medzi procesmi bežiacimi na tom istom počítači) pod jednotné programátorské rozhranie. Toto prácu zjednoduší a navyše umožní rozšírenie použiteľnosti aplikácie na akúkoľvek sieť na ktorej je implementovaný komunikačný protokol IP. Vďaka tomu bude jednoduché aplikáciu aj otestovať na rôznych bezdrôtových sieťach bez nutnosti jej úpravy.

Vytvorená aplikácia má v konečnej verzií umožňovať prenos obrazu medzi dvomi a viacerými počítačmi. Toto je treba mať na mysli pri návrhu ideológie aplikácie.

Vzhľadom na to sa javí výhodný systém distribúcie dát, pri ktorom počítač, ktorý chce po sieti sprístupniť obraz snímaný jeho snímacím zariadením bude vystupovať ako server, a každý pripojený klient bude môcť tento obraz prijímať a zároveň môže tiež vystupovať ako server – zdroj dát pre iných klientov.

V rámci problematiky komunikácie je treba ešte rozhodnúť aký komunikačný protokol bude použitý vo transportnej vrstve OSI modelu. Ako je uvedené v kapitole 2.3.1 je vhodné použiť TCP protokol, ktorý zabezpečuje, že prijaté dáta nie sú poškodené, ani sa žiadne dáta cestou nestratili, alebo nezduplikovali, a sú v správnom poradí. Aj keď tento protokol uberá z dátovej rýchlosti väčším množstvom režijných dát, mnohé funkcie, ktoré implementuje sú aj pre prenos obrazu nevyhnutné a bolo by ich treba aj pri použití iného protokolu (napr. UDP) implementovať.

4.2 Koncepcia aplikácie

Celý postup tvorby aplikácie určenej na prenos obrazu po sieti možno rozdeliť do troch základných častí:

- grafické rozhranie,
- snímanie obrazu,
- sieťová komunikácia.

Vzhľadom na skutočnosť, že bol zvolený programovací jazyk C je na programovanie aplikácie najvhodnejšie použiť vstavané programovacie rozhranie operačného systému *Microsoft Windows* nazývane *Windows API*, alebo *WinAPI*.

4.2.1 Windows API

Vychádzajúc z informácií uvedených v [7] a [10] je možné tvrdiť, že *Windows API* predstavuje základný súbor programovacích rozhraní (API) operačného systému *Microsoft Windows*. Niekedy tiež nazývané *Win32 API* môže byť rozdelené do ôsmych základných skupín služieb:

Základné služby – poskytujú prístup k základným funkciám, ktoré poskytuje operačný systém *Microsoft Windows*, čo zahŕňa prístup k súborom, zariadeniam, procesom a vláknam. Taktiež poskytujú možnosti spracovávania chýb vzniknutých počas behu aplikácie (napr. funkcia `GetLastError()`). V 32 bitovej verzií operačného systému sú tieto funkcie implementované v systémovej knižnici `kernel32.dll`.

Pokročilé služby – zabezpečujú prístup k prídavným funkciám *kernelu* operačného systému ako *Windows registry*, vypnutie, reštartovanie systému, správu služieb a používateľských kont. Tieto funkcie sa nachádzajú v súbore `advapi32.dll`.

Rozhranie grafických zariadení – umožňuje grafický výstup na monitor, tlačiarne a iné zariadenia. Toto rozhranie je implementované v knižnici `gdi32.dll`.

Služby grafických používateľských rozhraní – poskytujú vytváranie a správu grafických okien ako tlačidlá a iné prvky grafických rozhraní v operačných systémoch Windows. Taktiež umožňujú spracovávanie vstupu z klávesnice a myši. Základný vzhľad grafických prvkov je implementovaný v súbore `user32.dll`. Od verzie operačného systému *Windows XP* sú ovládacie prvky s pokročilým vizuálnym vzhľadom obsiahnuté v knižnici `comctl32.dll`.

Knižnica základných dialógových okien – súbor `comdlg32.dll` obsahuje set základných dialógových okien.

Knižnica ovládacích prvkov – základné a aj pokročilé ovládacie prvky, implementované v súbore `comctl32.dll`.

Windows Shell – poskytuje prístup k shell-u operačného systému

Sieťové služby – umožňujú prístup k sieťovým službám operačného systému ako *NetBIOS*, *Winsock*, *NetDDE*, *RPC*.

V nasledujúcich podkapitolách budú podrobnejšie rozobrané niektoré súčasti *Windows API* použité pri vytváraní aplikácie.

4.2.1.1 Threads of execution

Vlákno (*thread*) je základná jednotka vykonávajúca priradený kus spustiteľného kódu, ktorej operačný systém prideľuje výpočtový čas procesoru. Každý proces spúšťa minimálne jedno vlákno – hlavné vlákno, ktorého vstupný bod (adresa v pamäti počítača na ktorej sa začína spustiteľný kód určený na výkon daným vláknom) je zároveň vstupným bodom procesu (vo win32 je to funkcia s názvom `WinMain(...)`). Toto vlákno potom môže spúšťať ďalšie vlákna a určiť im ich vlastný vstupný bod.

Využívanie viacerých spustených vláken v rámci jedného procesu je užitočné najmä na systémoch, ktoré obsahujú viac ako jedno jadro procesora. Ale v situácií, kedy jedno vlákno musí napríklad čakať na dokončenie istej operácie majú veľký zmysel aj na jednojadrových systémoch.

Časové úseky (*time slice*) v ktorých môže vlákno vykonávať svoj kód prideľuje operačný systém v závislosti na mnohých parametroch. Operačný systém tiež sám preruší beh vlákna (bez jeho vedomia) aby mohol prideliť ďalší časový úsek inému vláknu. Toto správanie operačného systému sa nazýva *preemptívny multitasking*. Ak sa v systéme nachádza viac ako jedno jadro procesora, môže nastať situácia, kedy sú súčasne vykonávané dve a viac (podľa počtu jadier procesora) vláken, pričom tieto vlákna môžu prináležať jednému procesu.

Spolu s *multithreading*-om (súčasné využívanie viacerých aktívnych vláken) prichádzajú aj isté problémy, a to najmä v situácií, keď treba pristupovať do toho istého bloku pamäte z rôznych vláken. Keďže je takmer vždy vyžadované aby vlákna nejakým spôsobom spolu komunikovali, nastáva táto situácia veľmi často. Ak by nastala situácie, kedy sa dve vlákna pokúšajú pristupovať k jednému pamäťovému miestu, pričom aspoň jedno z nich do tohto miesta zapisuje, môže byť výkon tohto vlákna prerušený operačným systémom uprostred tejto operácie, keďže väčšinou nie je možné uskutočniť zápis do pamäte jednou inštrukciou procesora. Ak táto situácia nastane, môže sa stať, že iné vlákno prečíta takto porušené dáta a dochádza k chybe. Preto je nutné prístup vláken k zdieľanej pamäti synchronizovať, to znamená vylúčiť situáciu, kedy dve vlákna súčasne pristupujú do jednej pamäte. K tomuto účelu operačný systém poskytuje nástroje nazývané synchronizačné objekty.

Takýmto objektom je napríklad kritická sekcia (*Critical section*). Je to synchronizačný objekt, ktorý umožňuje označiť kus kódu, ktorý môže súčasne vykonávať len jedno vlákno. Princíp spočíva v tom, že objekt kritickej sekcie môže získať súčasne len jedno vlákno. Získanie kritickej sekcie je uskutočnené zavolaním funkcie `EnterCriticalSection(...)` s ukazovateľom na objekt kritickej sekcie ako parametrom. Ak jedno vlákno týmto spôsobom získa kritickú sekciu a iné vlákno sa tiež pokúsi o získanie tej istej kritickej sekcie, je jeho toto vlákno pozastavené dokým vlákno, ktoré vlastní kritickú sekciu ju neopustí. Vlákno opustí kritickú sekciu volaním funkcie `LeaveCriticalSection(...)`. Len čo vlákno, ktoré doteraz vlastnilo kritickú sekciu túto opustí, môže ju získať iné vlákno. Využitie kritických sekcií je výhodné a rýchle, keďže operačný systém ku každej kritickej sekcií udržuje informáciu o tom, ktoré vlákno kritickú sekciu vlastní, a preto ak nastane situácia, že by mal

prideliť výpočtový čas vláknu, ktoré čaká na získanie tejto kritickej sekcie, a teda nemôže pokračovať, pridelí tento výpočtový čas radšej vláknu, ktoré vlastní danú kritickú sekciu. Toto správanie je veľmi užitočné najmä v situáciách, keď na získanie jednej kritickej sekcie čaká veľký počet vláken.

Ďalším základným synchronizačným objektom je udalosť (*Event*). Udalosť, ako už napovedá názov, je synchronizačný objekt, ktorí slúži na signalizáciu programom definovaných udalostí. Udalosť môže nadobúdať dva stavy: signalizovaný a nesignalizovaný. Vlákno môže uskutočniť čakanie na určitú udalosť (volaním funkcie `WaitForSingleObject(...)`, ktorá umožňuje aj nastaviť *timeout*, teda čas po ktorom uplynutí vlákno ukončí čakanie, či už bol objekt signalizovaný, alebo nie), ak sa udalosť nachádza v nesignalizovanom stave, výkon vlákna je pozastavený, dokým nie je stav udalosti nastavený na signalizovaný. Ak počas čakania na udalosť dôjde k jej signalizácií a jedna sa o takzvaný *auto-reset event*, je ihneď stav tejto udalosti nastavená späť na nesignalizovaný stav. Pri *manual-reset evet*-och k tomuto nedôjde.

Existujú ešte ďalšie synchronizačné objekty, ktoré však vo vytváranej aplikácií nebudú potrebné a preto tu nie sú spomínané.

4.2.1.2 DirectShow

Multimediálny *framework* pre operačný systém *Microsoft Windows* s názvom *DirectShow* slúži na programové riadenie zachytávania, prehrávania a kódovania obrazu aj zvuku. Je založený na objektovom *frameworku* spoločnosti *Microsoft* s názvom *Component Object Model*.

Tento *framework* je najvhodnejší aj pre zachytávanie obrazu z kamery, aj navzdory skutočnosti, že existujú aj iné možnosti ako napríklad *Video for Windows*, ktoré je ale už zastarané a nie je naďalej podporované v moderných operačných systémoch.

4.2.1.3 Windows Sockets 2

Windows Sockets, ktoré sú založené na *Berkeley Sockets API* používané v operačných systémoch BSD, umožňujú vytváranie aplikácií komunikujúcich prostredníctvom počítačových sietí nezávisle na použitom prenosovom prostriedku. Taktiež umožňujú komunikáciu pomocou rôznych sieťových protokolov a po rôznych typoch sietí, avšak je nutné myslieť na skutočnosť, že pri použití rôznych komunikačných protokolov je nutné aj k používaniu socketov, a najmä dát prenesených pomocou týchto socketov, pristupovať odlišne. Príklad, ako bolo spomenuté v kapitole 2.3.1, pri použití komunikačného protokolu TCP/IP nie je potrebné sa starať o obnovenie poradia správ alebo o vyžiadanie preposlania stratených, či porušených dát, keďže o tieto operácie sa stará samotný protokol. Naopak dátové správy posielaná protokolom UDP nemusia zachovať svoje poradie a nie je ani zaručené, či správa naozaj dorazí, alebo či nedorazí tá istá správa viac ako jeden krát.

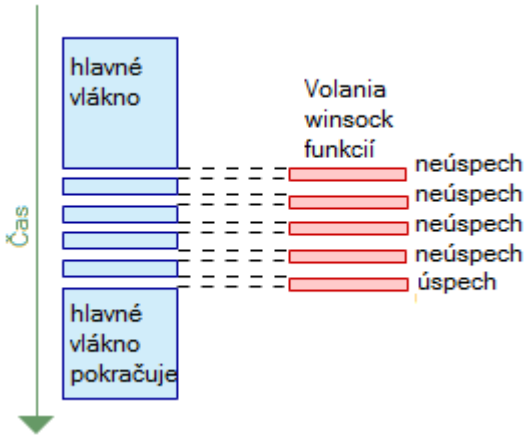
Programovacie rozhranie *Windows Sockets API* v predvolenom stave používajú na volanie funkcií slúžiacich na odoslanie alebo prijatie dát takzvaný blokovací (alebo tiež synchrónny) režim. V tomto režime je pri každom funkčnom volaní týchto funkcií

pozastavený výkon volajúceho vlákna dokým nie je vstupne/výstupná (IO) operácia dokončená. Takéto použitie je samozrejme vylúčené v aplikácií, ktorá má spustené len jedno vlákno a toto vlákno zobrazuje aj grafické rozhranie, a to z dôvodu, že výkon tohto vlákna nemôže byť prerušený, keďže musí obsluhovať vstupy od užívateľa a operačného systému. Výsledok zablokovania výkonu tohto vlákna by bol stav aplikácie, kedy táto nereaguje na žiadny užívateľský vstup – aplikácia „zamrzne“.

Z tohto dôvodu je nutné použiť nejaký postup, ktorý zabezpečí, že k tomuto stavu nedôjde. *Windows Sockets* poskytujú riešenie tohto problému v podobe takzvaného neblokovacieho režimu, kedy každé funkčné volanie funkcií slúžiacich na vykonanie určitej vstupne/výstupnej operácie (poslať, prijať dáta, pripojiť k serveru, prijať pripojenie klienta) môže byť:

- dokončené ihneď (či už úspešne, alebo s chybou),
- neúspešné s tým, že funkcia vráti chybu informujúcu o skutočnosti, že *framework* by v tomto momente zablokoval priebeh volajúceho vlákna. V tomto prípade je nutné operáciu zopakovať niekedy v budúcnosti znova.

Metód ako zistiť kedy sa začatá neblokovacia operácia ukončila je niekoľko. Tieto metódy sa nazývajú *I/O modely*. Najjednoduchší a asi najhorší *I/O model* je neustále cyklické opakovanie operácie dokým sa zvolená operácia úspešne nedokončí (*Polling*). Priebeh takéhoto programu je znázornený na Obr. 4.1. Veľkou nevýhodou tohto prístupu je veľká neefektivita využitia procesora počítača, keďže program musí prechádzať slučkou, dokým sa operácia úspešne neskončí. Naproti tomu v blokovacom režime nevyužíva program čakajúci na dokončenie vstupne/výstupnej operácie takmer žiadny procesorový čas. Jedinou výhodou oproti blokovaciemu režimu je skutočnosť, že vo vnútri tejto slučky nadobúda program kontrolu a môže tak napríklad spracovávať vstupy od užívateľa.

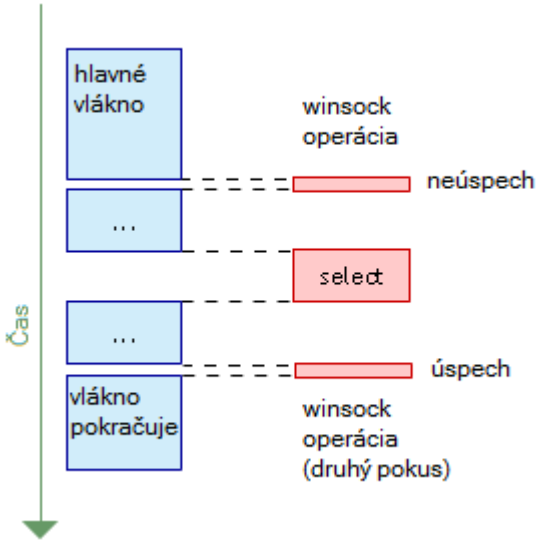


Obr. 4.1 „Polling“ metóda

Ďalší možný prístup neblokovacieho režimu je použitie API funkcie `select()`. Tento spôsob sa veľmi podobá na blokovací režim, keďže po prvom neúspešnom pokuse o okamžité vykonanie vstupne/výstupnej operácie je zavolaná funkcia `select()`, ktorá zablokuje výkon vlákna dokým nebude možné operáciu úspešne zopakovať. Oproti blokovaciemu režimu má tento postup výhodu v tom, že môže čakať na pripravenosť viacerých socketov naraz. Možný priebeh programu s využitím funkcie *select* je na Obr. 4.2.

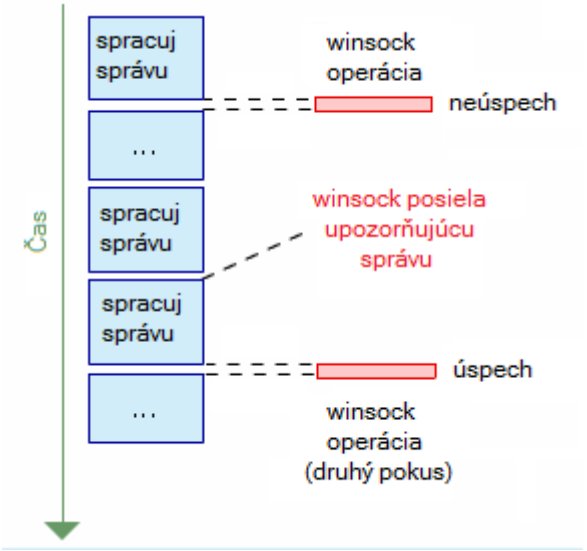
Takmer všetky programy pre operačný systém *Microsoft Windows* vytvárajú niektoré druhy okien. Programovacie rozhranie *Windows Sockets* umožňujú pomocou správ doručovaných systémom každému oknu upozorniť na dokončenie určitej vstupne/výstupnej operácie.

Obr. 4.3 znázorňuje takéto riešenie.



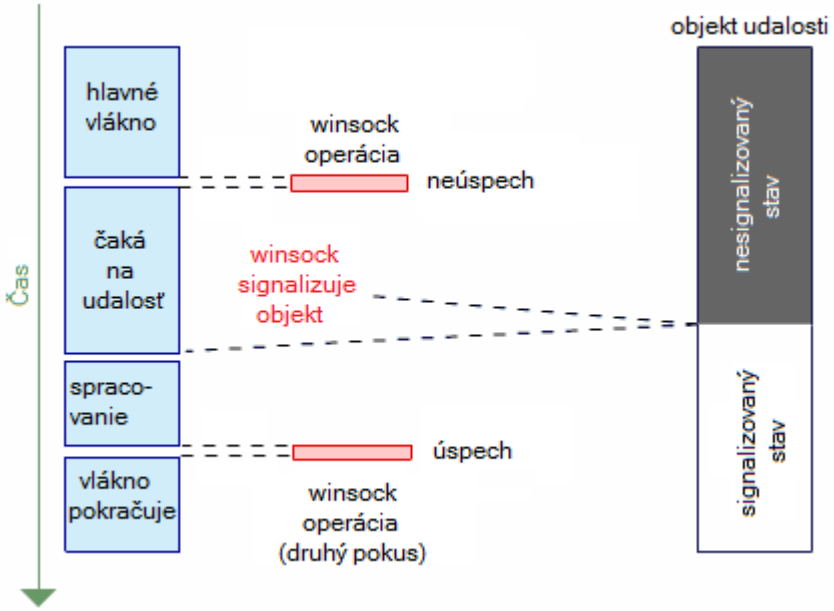
Obr. 4.2 Metóda s využitím funkcie select

Toto riešenie predstavuje relatívne jednoduchý a pomerne efektívny spôsob realizácie neblokovacieho režimu socketov. Jedinou nevýhodou je skutočnosť, že systém doručovania správ oknám aplikácie nie je dostatočne rýchli a preto by nemal byť tento spôsob použitý pre servery s viac ako 1000 pripojenými užívateľmi. Taktiež je tu nutnosť vytvorenia aspoň jedného okna aplikáciou, čo môže byť nevhodné napríklad pre servery bežiace ako služby operačného systému.



Obr. 4.3 Neblokovací režim pomocou oknových správ

Podobné riešenie ako pri použití oknových správ operačného systému ponúka postup s využitím objektov udalostí. Tento spôsob neblokovacieho režimu je rýchlejší ako použitie oknových správ a umožňuje lepšie oddelenie sieťového kódu od ostatného kódu. Toto riešenie je popísané na Obr. 4.4.



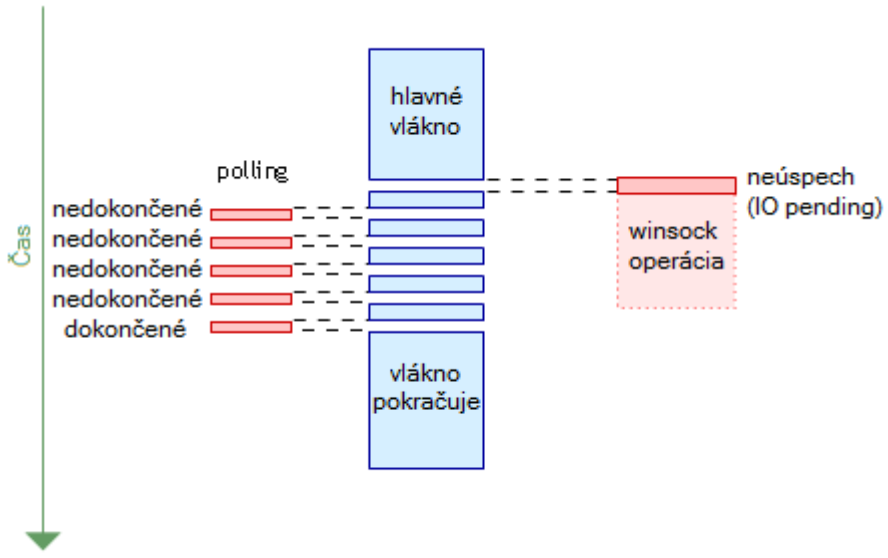
Obr. 4.4 Neblokovací režim s využitím objektu udalosti

4.2.1.4 Windows Sockets a Overlapped IO

Overlapped IO je veľmi efektívny IO model. Na rozdiel od asynchrónnych IO modelov, spomínaných doteraz, *Overlapped IO* neinformuje o zmene sieťových udalostí, ako prišli dáta, možno posielať dáta, ale upozorňuje na dokončenie vstupne/výstupnej operácie. To znamená že po prvom neúspešnom neblokovacom volaní socketovej opeácie nie je nutné skúšať tú istú operáciu vykonať znova, ale stačí počkať dokým *Winsock framework* neinformuje program o dokončení vstupne/výstupnej operácie. Spôsoby akými *Winsock* informuje program o dokončení operácie môžu byť rôzne, niektoré z nich budú podrobnejšie prebrané ďalej.

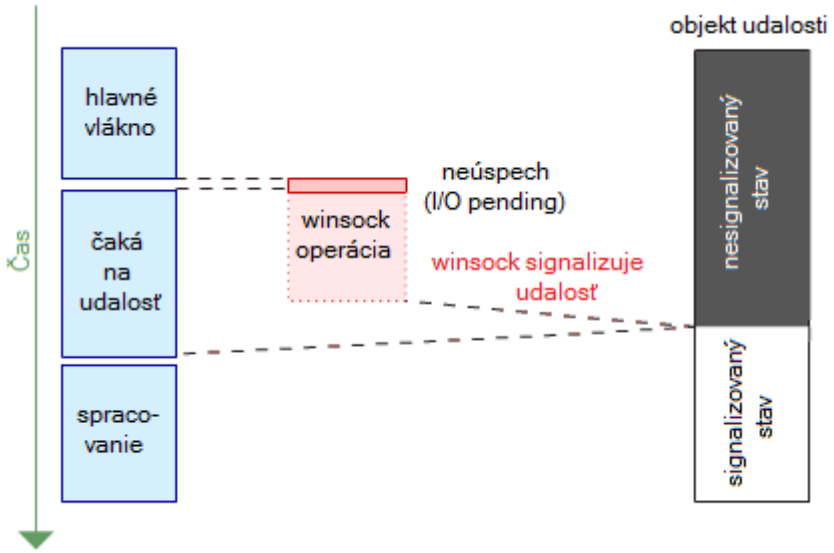
Nevýhodou tohto prístupu je komplikovanosť implementácie a tiež skutočnosť že tento IO model je podporovaný len v operačných systémoch Windows NT a vyšších. Z čoho vyplýva, že takýto program nebude možné spustiť na operačných systémoch Windows 9x/ME. Toto však nie je veľká prekážka, keďže tieto operačné systémy sa dnes už takmer nepoužívajú.

Rovnako ako u asynchrónnych modelov aj u *Overlapped IO* je možné použiť takzvaný *polling* prístup, teda cyklické testovanie, či sa vstupne/výstupná operácia nedokončila. Znázornenie behu programu realizujúceho takýto IO model je na Obr. 4.5.



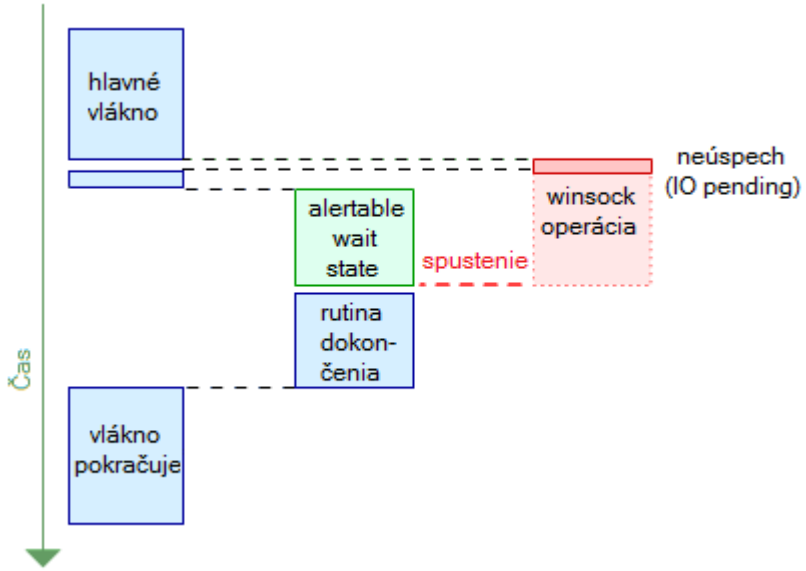
Obr. 4.5 Overlapped IO metódou „polling“

Podobným prístupom ako pri asynchrónnych modeloch je možné použiť aj objekt udalosti, ktorú Winsock nastaví do signalizovaného stavu po tom ako je zvolená operácia dokončená. Tento prístup je znázornený na Obr. 4.6.



Obr. 4.6 Overlapped IO s použitím udalostí

Ďalšou možným *Overlapped IO* modelom je model s použitím rutín dokončenia, čo sú funkcie, ktoré sú vykonané operačným systémom v okamihu dokončenia vstupne/výstupnej operácie. Táto funkcia je vykonaná v kontexte vlákna, ktoré iniciovalo vstupne/výstupnú operáciu pomocou mechanizmu zvaného *Asynchronous Procedure Call* (APC). APC je čosi ako vynútenie vykonania určitej funkcie počas behu vlákna tak aby táto funkcia mohla byť vykonaná a po jej dokončení mohlo vlákno ďalej pokračovať v pôvodnej operácií. Každé vlákno má svoju APC frontu, kde sú uložené APC čakajúce na vykonanie. Na to aby systém mohol vykonať funkciu čakajúcu v APC fronte je nutné aby vlákno bolo v takzvanom pohotovostnom čakacom režime (*alertable wait state*), to je kvôli tomu, že systém nemôže prerušiť výkon vlákna v ľubovoľnom bode. Do tohto stavu sa vlákno dostane pri uspaní, alebo čakaní na inú udalosť. Obr. 4.7 znázorňuje tento *Overlapped IO* model.

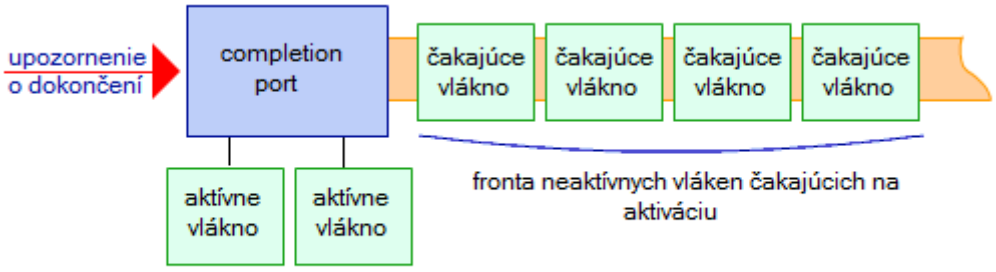


Obr. 4.7 Overlapped IO s využitím dokončovacej procedúry

Posledným a zrejme najefektívnejším spôsobom komunikácie pomocou Overlapped IO je model s použitím dokončovacích portov (*Completion ports*). Tento model bude podrobnejšie rozobraný v nasledujúcej kapitole.

4.2.1.5 Completion ports

Completion port (Obr. 4.8) je mechanizmus dostupný v jadre operačných systémov *Microsoft Windows NT* (staršie verzie tento mechanizmus nepodporujú) umožňujúci veľmi efektívnu správu vláken slúžiacich na spracovávanie správ o dokončení (*Completion packets*) komunikačného modelu založeného na modely *Overlapped IO*.



Obr. 4.8 Dokončovací port

Ideológia stojaca za mechanizmom dokončovacích portov je nasledujúca. Po vytvorení dokončovacieho portu je možné s týmto objektom asociovať viacero socketov (prípadne objektov typu *file handle*). Každému objektu asociovanému s dokončovacím portom je možné priradiť takzvaný *Completion Key*, čo môže byť napríklad ukazovateľ

na dátovú štruktúru obsahujúcu informácie o spojení, ktoré je reprezentované daným soketom. *Completion port* obsahuje dva fronty:

1. front príchodzích správ o dokončení (*completion packet*),
2. front neaktívnych vláken určených na spracovávanie udalostí signalizujúcich dokončenie prekrývajúcej sa (*overlapped*) vstupne/výstupnej operácie.

Prvý z týchto front je typu FIFO, keďže je samozrejmé, že prvá príchodzia správa má byť spracovaná najskôr. Druhý je typu LIFO, a to práve z toho dôvodu, aby vlákno, ktoré práve dokončilo spracovávanie operácie neprešlo do čakacieho stavu, pokiaľ sú v prvej fronte ešte nejaké správy. Toto opatrenie je tu vykonané preto, že uspávanie a preberanie vlákna zahrňuje istú réžiu, ktorá by bola v tomto prípade zbytočná.

Pri dokončení *overlapped IO* operácie na objekte (*socket* alebo *file handle*) asociovanom s dokončovacím portom je odoslaná správa nazývaná *completion packet* do tohto portu. Ten potom túto správu doručí jemu priradeným vláknam na spracovanie.

Každý *completion port* musí mať priradené jedno alebo viacej vláken, ktoré spracovávajú správy o dokončení. Vlákno je k dokončovaciemu portu priradené prvým zavolaním funkcie slúžiacej na výber správ o dokončení z frontu. Ak je front prázdny, vlákno sa v tomto funkčnom volaní zablokuje a je prebrané až v okamihu, keď dokončovací port rozhodne, že existuje správa o dokončení, ktorú má toto vlákno spracovať. Vlákno zostáva asociované s dokončovacím portom až do jeho ukončenia, alebo do zavolania funkcie na vybranie správy o dokončení iného dokončovacieho portu. Z toho vyplýva skutočnosť, že každé vlákno môže byť v jednom okamihu asociované len s jedným CP (*completion port*).

Vhod prichádza otázka: aký je optimálny počet vláken asociovaných s jedným CP? Podľa [11] existujú dva optimálne prístupy k riešeniu tejto otázky a to:

- dynamicky sa rozrastajúce pole vláken (*thread pool*) podľa aktuálnych potrieb,
- statický počet vláken, ktorý je určený ako dvojnásobok jadier procesorov nachádzajúcich sa v počítači, na ktorom aplikácia beží.

Druhý z týchto dvoch možných prístupov je podstatne jednoduchší a poskytuje porovnateľné výsledky.

S počtom priradených vláken súvisí aj parameter dokončovacieho portu, nastavovaný pri jeho vytváraní, nazývaný počet súbežných vláken (*Number Of Concurrent Threads*). Tento parameter určí koľko vláken z fronty neaktívnych vláken môže byť aktivovaných naraz. Ak by mal počet aktívnych vláken dokončovacieho portu narásť nad hodnotu tohto parametru nebude už ďalšie vlákno z fronty neaktívnych vláken aktivované, aj keby vo fronte správ o dokončení čakala nespracovaná správa. Predvolená hodnota pre tento parameter je dvojnásobok počtu procesorových jadier nachádzajúcich sa v systéme. Predvolená hodnota tohto parametru bola odvodená z úvahy, že nemá príliš zmysel aby bolo naraz spustených a aktívnych viac vláken ako počtu jadier procesoru, avšak pri spracovávaní jednotlivých správ o dokončení môže pracujúce vlákno prejsť do neaktívneho stavu napríklad uskutočnením synchrónnej vstupne/výstupnej operácie alebo čakaním na istú udalosť. Preto bol ako optimálny

kompromis zvolený počet súbežných vláken ako dvojnásobok počtu procesorových jadier v počítači. Taktiež je samozrejmé, že nemá veľký zmysel priraďovať k dokončovaciemu portu viac vláken ako je hodnota tohto parametru, keďže by potom existovali neaktívne vlákna aj pri extrémnom zaťažení dokončovacieho portu (veľký počet prichádzajúcich *completion packetov*), a ako bolo spomínané v kapitole 4.2.1.1 veľký počet neaktívnych vláken môže spôsobiť spomalenie behu programu, keďže operačný systém musí „hľadať“ vlákna, ktorým by mohol priradiť výpočtový čas procesoru.

Completion Ports taktiež umožňujú programu posielať vlastné správy o dokončení na dokončovací port. Tento mechanizmus môže byť využitý napríklad pri odstraňovaní dokončovacieho portu, keď je potrebné ukončiť všetky priradené vlákna.

Avšak systémové prostriedky, ktoré zaberá CP nie sú uvoľnené dokým nie sú odstránené všetky referencie na CP. Referencia na dokončovací port je každý objekt asociovaný s CP a tiež jeho vlastný deskriptor (*handle*). To znamená, že aby operačný systém uvoľnil systémové prostriedky priradené objektu dokončovacieho portu je nutné ukončiť všetky spojenia, ktoré pomocou tohto portu realizovali komunikáciu a aj samotný deskriptor dokončovacieho portu volaním API funkcie `CloseHandle()`.

4.3 Realizácia aplikácie

Veľké logické celky vytváranej aplikácie spomínané na začiatku kapitoly 4.2 je možno rozdeliť ešte do menších celkov. Každý tento menší celok je v zdrojovom kóde reprezentovaný samostatným párom hlavičkového a zdrojového súboru (*.h a *.c).

Sú to tieto celky:

- grafické rozhranie
  - FrameWnd
  - ChildWnd
  - Dialogs
  - VideoWnd
- snímanie obrazu
  - FrameGrabber
  - VideoInputC
- sieťová komunikácia
  - Buffer
  - Networking
  - Protocol
  - Server
- spoločné
  - Common
  - List
  - CTWMD
  - MemDbg

- ReadersWriterLock
- WVFrame

Tieto celky budú podrobne popísane v nasledujúcich podkapitolách.

4.3.1 Spoločné

Do tejto skupiny patria bloky, ktoré zdieľa viacero rôznych častí. Najzákladnejším blokom je `Common`.

4.3.1.1 Common

Tento blok pozostáva len z hlavičkového súboru `Common.h` (zdrojový súbor `Common.c` je prázdny), obsahom ktorého sú definície všetkých hlavičkových súborov operačného systému, ktoré sa budú používať, ďalej sú tu nadefinované všetky dátové štruktúry a makrá používané inými blokmi. Taktiež sa tu nachádzajú deklarácie globálnych premenných zdieľaných medzi blokmi.

Hlavičkový súbor `Common.h` je nadefinovaný v každom bloku v celom projekte.

4.3.1.2 List

Ako už názov napovedá jedná sa o implementáciu jednosmerne viazaného zoznamu, v ktorom každý prvok je reprezentovaný všeobecným ukazovateľom, preto môže tento zoznam obsahovať akýkoľvek typ dát.

Zoznam je vytvorený volaním funkcie `CreateList()`, ktorá má jeden argument `lpEqFcn`, čo je porovnávacia funkcia priradená k zoznamu slúžiaca na zistenie, či sa dva prvky zoznamu rovnajú. Táto funkcia je použitá pri vymazávaní prvku zo zoznamu, keďže samotný zoznam nevie určiť, kedy dva ukazovatele ukazujú na rovnaký objekt. Ak je táto hodnota `NULL`, je rovnosť dvoch objektov určená na základe rovnosti ukazovateľov.

V tomto bloku sú implementované funkcie určené na vykonávanie základných operácií so zoznamom vrátane funkcií určených na enumeráciu všetkých prvkov zoznamu a zmazanie zoznamu.

Špeciálnu funkciu plní funkcia `CreateItem(...)`, ktorá slúži na pridanie prvku do zoznamu, ktorý je určený ukazovateľom `lpLs` a to takým spôsobom, že zároveň alokuje dostatočné miesto ja pre dáta prvku zoznamu, ktorý sa v pamäti bude nachádzať hneď za štruktúrou definujúcou tento prvok. Veľkosť pamäte potrebnej na obsiahnutie týchto dát je funkcií predaná pomocou parametru `cItemSize`. Toto opatrenie je implementované, kvôli zmenšeniu fragmentácie pamäte, ktorá vzniká pri alokovaní veľkého počtu malých blokov pamäte. Treba tu však mať na pamäti, že po odstránení takéhoto prvku zo zoznamu sa uvoľní aj pamäť obsahujúca dáta tohto prvku. Toto správanie však v mnohých prípadoch nie je problém.

Všetky funkcie pracujúce so zoznamom sú vláknovo bezpečné, to znamená, že prístup k objektu zoznamu je synchronizovaný a pre prácu so zoznamom nie je nutné vykonávať synchronizáciu explicitne.

4.3.1.3 MemDbg

Jedná sa o nástroj určený len na *debugovanie* aplikácie. Jeho použitie pri preklade aplikácie je v hlavičkovom súbore `Common.h` podmienené *debugovacou* konfiguráciou prekladu aplikácie. Vo verzií *Relase* nie je tento blok použitý.

Tento blok nahradzuje niektoré funkcie slúžiace na dynamickú správu pamäte vlastnými funkciami, ktoré okrem vykonania pôvodnej operácie danej funkcie udržujú zoznam všetkých alokovaných blokov pamäti. Pri ukončení aplikácie tento blok vypíše zoznam všetkých neuvoľnených blokov pamäte do *debugovacieho* výstupu.

4.3.1.4 CTWMD

Tento blok zabezpečuje doručovanie oknových správ oknám z iných vláken, ako z vlákna, ktoré okno vytvorilo. V princípe je toto zabezpečené tak, že pri vytvorení objektu CTWMD funkciou `CreateCTWMD()` vo vlákne, ktoré vytvára okná, ktorým je nutné doručovať správy z iných vláken, je zaregistrovaný takzvaný *Windows Hook*, ktorý odchytáva špeciálne druhy správ posielané vláknu funkciou `SendCTMessage(...)` z iných vláken, ktorá zároveň počká na odpoveď daného okna a túto vráti.

4.3.1.5 ReadersWriterLock

V tomto bloku je realizovaný objekt slúžiaci na efektívnu synchronizáciu vláken v situácií, kedy existuje mnoho vláken, ktoré chcú z danej chránenej pamäti len čítať (čo môžu robiť súčasne) ale len jedno vlákno, ktoré chce na danú pamäťovú pozíciu zapisovať. Teda pri distribúcií nejakej informácie viacerým vláknam.

Tento objekt môže vlákno zamknúť dvomi spôsobmi: s úmyslom do chránenej pamäte zapisovať (funkcia `WriterEnter(...)`) alebo s úmyslom len čítať z tejto pamäte (funkcia `ReaderEnter(...)`). V druhom prípade je vláknu dovolené pokračovať, pokiaľ sa iné vlákno nepokúša získať zapisovací zámok objektu, teda pokiaľ iné vlákno nezavolalo funkciu `WriterEnter(...)`. V tomto prípade je vlákno pokúšajúce sa získať čítací zámok objektu pozastavené, dokým zapisujúce vlákno neuvoľní zapisovací zámok objektu. Vlákno pokúšajúce sa získať zapisovací zámok objektu je pozastavené v prípade, že existujú vlákna, ktoré majú čítací zámok objektu. Ale keďže po zavolaní funkcie `WriterEnter(...)` nie je povolené iným vláknam získať čítací zámok objektu, časom všetky vlákna ktoré mali čítací zámok ho uvoľnia a potom je povolené zapisovaciemu vláknu pokračovať. Ďalšie vlákna nemôžu získať čítací zámok objektu, dokým zapisovacie vlákno neuvoľní jeho zámok.

4.3.1.6 WVFrame

V tomto bloku sa nachádzajú funkcie na správu dátových štruktúr slúžiacich na ukladanie snímok z kamery do pamäti počítača. Sú tu implementované funkcie slúžiace na alokovanie a uvoľňovanie pamäti pre snímok, jeho kopírovanie a zbalenie a rozbalenie hlavičiek snímok z/do byteového reťazca, ktorý je možné preniesť sieťou.

4.3.2 Grafické rozhranie

V tejto skupine blokov sa nachádzajú bloky vytvárajúce prvky grafického rozhrania aplikácie.

4.3.2.1 FrameWnd

V tomto bloku sa nachádza vstupný bod procesu. Po spustení aplikácie sú zaregistrované všetky potrebné triedy okien a vykonané iné operácie potrebné na úspešné spustenie aplikácie. Nakoniec je vytvorené hlavné okno aplikácie a program vojde do hlavnej slučky oknových správ.

Hlavné okno je riešené štýlom *Multiple Document Interface*, kde každé okno „dokumentu“ predstavuje jedno pripojenie na server poskytujúci obrazové dáta, ktoré budú do tohto okna vykresľované.

Taktiež je tu definovaná oknová rocedúra hlavného okna.

4.3.2.2 ChildWnd

Tu je definovaná oknová procedúra okna, ktoré predstavuje dokument v *Multiple Document Interface*, ktoré tvorí hlavné okno aplikácie. A tiež funkcia vytvárajúca nové okno dokumentu.

4.3.2.3 Dialogs

Blok Dialogs obsahuje funkcie na vytváranie dialógových okien používaných v aplikácií.

4.3.2.4 VideoWnd

Tento blok slúži na zaregistrovanie triedy okna, ktoré je schopné na svoj povrch vykresľovať snímky typu WVFRAME, ktoré boli spomínané v kapitole 4.3.1.6. Okno tejto triedy tvorí celú plochu okna opísaného v kapitole 4.3.2.2.

4.3.3 Snímanie obrazu

Snímanie obrazu z kamery je realizované pomocou knižnice *VideoInput*, ktorú využíva napríklad aj knižnica *OpenCV*. V podstate sa jedná o knižnicu, ktorá zjednodušuje prácu s *frameworkom DirectShow*.

4.3.3.1 VideoInputC

Jedná sa o *wrapper* objektovo orientovaného rozhrania knižnice *VideoInput*. Tento blok k funkciám tejto knižnice väčšinou nepridáva žiadnu inú funkcionalitu, len umožňuje vytvorenie, zmazanie a volanie metód objektu typu *VideoInput* z kódu napísaného v jazyku C. Jedinú pridanú funkcionalitu má funkcia slúžiaca na získanie mena snímacieho zariadenia, ktorá zároveň prekladá názov zariadenia z ANSI reťazca do UNICODE formátu.

4.3.3.2 FrameGrabber

Tento blok spúšťa a spravuje vlákno slúžiace na snímanie snímok z kamery do pamäti počítača. Tento snímok je v pamäti počítača popísaný dátovou štruktúrou VWFRAME. Keďže sa predpokladá, že zosnímané snímky bude chcieť iné vlákno spracovávať je tento snímok chránený synchronizačným objektom *ReadersWriterLock* spomínaným v kapitole 4.3.1.5.

Sú tu funkcie umožňujúce ako nakonfigurovanie a spustenie snímacieho zariadenia, tak aj zastavenie snímania.

Po nakonfigurovaní snímacieho zariadenia sa ihneď spúšťa snímanie obrazu z kamery do pripraveného objektu typu VWFRAME. Snímanie sa opakuje automaticky v nastavených časových intervaloch.

4.3.4 Sieťová komunikácia

Ako bolo spomínané v predošlých kapitolách na realizáciu sieťovej komunikácie budú použité sokety. So soketmi je však možné pracovať v rôznych IO modeloch, ktoré boli spomínané v kapitolách 4.2.1.3, 4.2.1.4 a 4.2.1.5.

Z týchto spomínaných prístupov bol vybraný práve model s využitím *Overlapped IO* a dokončovacích portov (*Completion ports*). Dôvodom prečo bol vybraný práve tento spôsob správy vstupne/výstupných operácií je, že sa jedná o najefektívnejší IO model.

Na realizáciu sieťovej komunikácie boli navrhnuté nasledujúce bloky:

4.3.4.1 Buffer

Spravuje pamäť alokovanú na pre účely sieťovej komunikácie. Taktiež zjednodušuje prijímanie a odosielanie blokov dát pomocou funkcií `WSASend(...)` a `WSARecv(...)` patriacich do *Winsock API*.

Keďže ani pri použití komunikačného protokolu TCP/IP nie je zaručené, že v jednom funkčnom volaní bude prijatý alebo odoslaný celý *buffer* vystavený spomínaným funkciám. Avšak je vždy vyžadované odoslanie alebo prijatie celého bloku dát, preto je nutné niekedy posielať dáta na viac krát. Za týmto účelom je tu implementovaná funkcia `JobDone(...)`, ktorá má vstupné parametre ukazovateľ na objekt BUFFER, počet bajtov prijatých alebo odoslaných v poslednej dokončenej operácií s týmto *bufferom* a maximálny počet bajtov prenesených v jednom volaní komunikačných funkcií. Ak je na odoslanie alebo prijatie celého *bufferu* nutné ďalšie volanie funkcie `WSASend(...)` alebo `WSARecv(...)` naplní táto funkcia správnymi hodnotami prvky dátovej štruktúry WSABUF, ktorá je súčasťou dátovej štruktúry BUFFER a ktorá potom môže byť priamo použitá pri volaní funkcií `WSASend(...)` alebo `WSARecv(...)`. Ak už bol odoslaný alebo prijatý celý *buffer* funkcia vráti hodnotu TRUE, indikujúc tak, že už nie je potrebné žiadne ďalšie volanie komunikačných funkcií `WSASend(...)` alebo `WSARecv(....)`.

4.3.4.2 Networking

Blok *Networking* implementuje jadro sieťovej komunikácie vytváranej aplikácie. Keďže ako sa spomína na začiatku kapitoly 4.3.4 na realizáciu komunikácie je využitý model *Overlapped IO* a *Completion ports*, je tu vytvorená správa dokončovacieho portu (*completion port*). Jeho vytvorenie je uskutočnené zavolaním funkcie `CreateIocpServer()`, ktorá alokuje a pri úspechu vráti ukazovateľ na dátovú štruktúru, ktorá obsahuje informácie o danom dokončovacom porte. V tejto dátovej štruktúre sa tiež nachádza ukazovateľ na zoznam v ktorom sa udržuje zoznam všetkých spojení asociovaných s daným dokončovacím portom pomovou funkcie `AddConnection(...)`. Funkcia `CreateIocpServer()` taktiež vytvorí a asociuje s CP vlákna určené na spracovávanie správ o dokončení (*completion packets*) posielaných na tento *completion port*.

Už pripojený a funkčný soket je možné asociovať s dokončovacím portom volaním funkcie `AddConnection(...)`, ktorá alokuje a pri úspechu vráti ukazovateľ na štruktúru typu CONNECTION. Pri neúspechu vráti funkcia hodnotu NULL. Funkcia alokuje o zadaný počet bajtov cez parameter `cDataSize` viac pamäte, do ktorej môžu byť uložené akékoľvek dáta, ktoré môžu napríklad obsahovať informácie potrebné pre komunikačný protokol realizovaný pomocou daného spojenia. Ukazovateľ na túto pamäť je možné získať makrom `GetProtocolDataPtr(...)`, ktoré prijíma parameter ukazovateľ na štruktúru CONNECTION popisujúcu dané spojenie. Ďalším parametrom funkcie `AddConnection(...)` je `lpProc`, čo je ukazovateľ na funkciu, ktorá bude volaná pri vzniknutí rôznych udalostí na danom spojení, ako napríklad vytvorenie spojenia, ukončenie odosielania či prijímania bloku dát, vzniknutie chyby a prerušenie spojenia. Posledné dva parametre tejto funkcie predstavujú adresu a port druhého účastníka komunikácie.

Ďalej sú tu funkcie `SendBuffer(...)` a `RecvBuffer(...)` realizujúce odoslanie alebo prijatie celého buffera, ktorý je funkciám odovzdaný ako parameter. O dokončení tejto operácie je informovaná procedúra protokolu, ktorá patrí ku spojeniu, ktoré bolo použité na odoslanie alebo prijatie dát.

Na korektné ukončenie spojenia je nutné najskôr zavolať funkciu `BeginDisconnect(...)` s ukazovateľom na štruktúru, ktorá popisuje dané spojenie, ako parametrom. Následne je nutné počkať, kým na túto skutočnosť zareaguje druhá strana, čo je realizované zavolaním funkcie `WaitForDisconnect(...)`. Keď už je spojenie korektne ukončené, je možné uvoľniť sieťové prostriedky, ktoré boli týmto spojením využívané, zavolaním funkcie `DestroyConnection(...)`.

4.3.4.3 Protocol

Protocol je blok obsahujúci procedúry protokolov využívané blokom Networking pre serverové a klientské spojenie. Serverový protokol zabezpečuje skopírovanie aktuálnej snímky vygenerovanej blokom FrameGrabber do svojej vlastnej pamäte, odoslanie

hlavičky snímku a následne aj obrazových dát snímku. Tento postup sa opakuje automaticky dokola, dokým sa spojenie neodpojí.

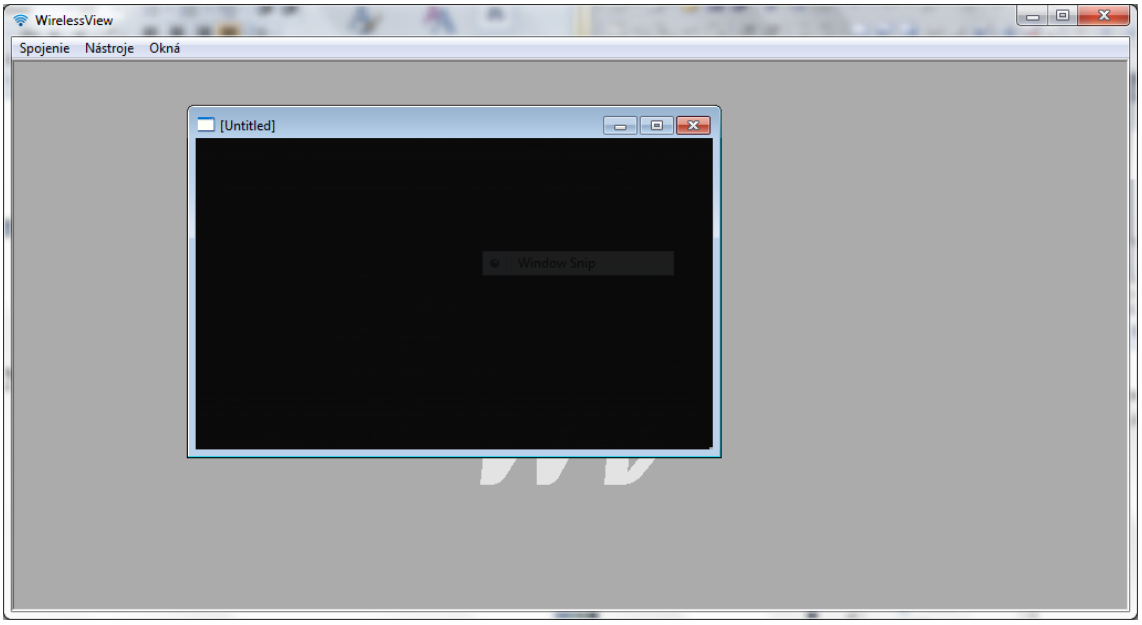
Na klientskej strane spojenia je najskôr prijatá hlavička snímku, tá je rozbalená a potom je v prípade potreby alokovaná pamäť dostatočne veľká aby obsiahla celú snímku. Potom sú do tejto pamäte prijaté obrazové dáta reprezentujúce snímku. Tento postup je zase opakovaný, dokým sa jedna z komunikačných strán nerozhodne spojenie ukončiť.

4.3.4.4 Server

Blok Server umožňuje vytvorenie serverového soketu, čakajúceho na príchodzie klientské spojenia, prijímanie klientských spojení a ich asociáciu s dokončovacím portom, pričom im priradí procedúru protokolu serverovej časti.

5 PREZENTÁCIA RIEŠENIA

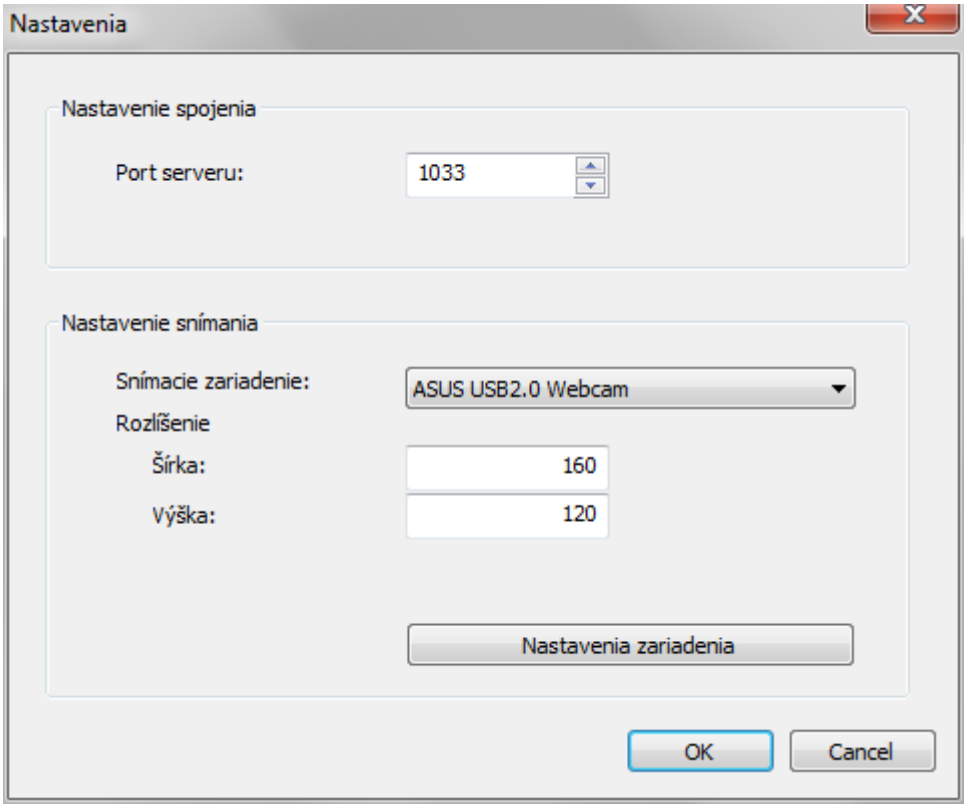
Grafické rozhranie vytvorenej aplikácie je znázornené na Obr. 5.1. Z obrázku je vidno, že grafické rozhranie hlavného okna je riešené vo forme *Multiple Document Interface*, pričom každé pripojenie na server je reprezentované v podobe jedného samostatného okna.



Obr. 5.1 Grafické rozhranie aplikácie

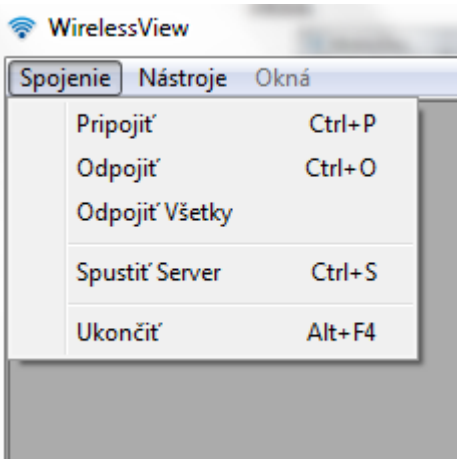
Ideológia aplikácie je taká, že počítač, ktorý chce sprístupniť obraz iným užívateľom po sieti, musí najskôr nakonfigurovať a spustiť server. Používateľ ktorí chce tento obraz prijímať sa musí na tento server pripojiť ako klient.

Dialógové okno konfigurácie servera (Obr. 5.2) je možné otvoriť cez hlavnú ponuka voľbou Nástroje>Nastavenia. V tomto okne je možné nastaviť TCP port na ktorom bude server očakávať prichádzajúce spojenia, ďalej je tu možné vybrať snímacie zariadenie a nastaviť rozlíšenie snímok snímaných zvoleným zariadením a posielaných po sieti klientom. Všetky zmeny vykonané v tomto okne sa pred ukončením aplikácie uložia do *databázy registry* aby zostali zachované pre nasledujúce spustenie aplikácie.



Obr. 5.2 Okno konfigurácie servera

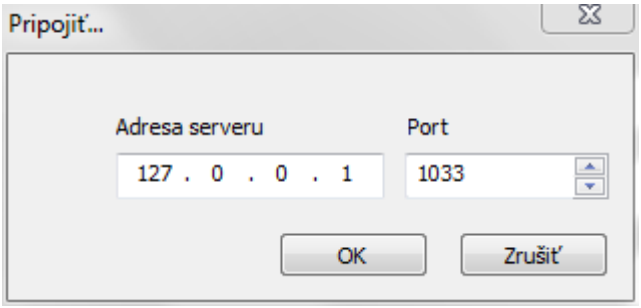
Po nakonfigurovaní serveru je ho možne spustiť cez ponuku Spojenie>Spustiť Server (Obr. 5.3). Po kliknutí na túto položku sa vytvorí *completion port*, pokiaľ už neexistuje, potom sa spustí zvolené snímacie zariadenie a začne snímať. Túto operáciu vykonáva blok *FrameGrabber* (kapitola 4.3.3.2). Blok *Server* (4.3.4.4) vytvorí načúvací soket s nastaveným zadaným portom a začne čakať na príchodzie požiadavky o pripojenie. Spomínaný *completion port* sa vytvára v bloku *Networking* (4.3.4.2) a keďže je využívaný aj pre klientské spojenia jeho vytvorenie mohlo byť inicializované aj pokusom o pripojenie k serveru.



Obr. 5.3 Ponuka Spojenie

Ak je raz už *completion port* vytvorený, neukončuje sa ani ak by už neexistovali žiadne jemu priradené spojenia. Jeho ukončenie je vykonané až pri ukončení aplikácie.

Z ponuky Spojenie je možné iniciovať aj klientské pripojenie k serveru, a to voľbou Pripojiť. Po kliknutí na túto položku sa objaví dialógové okno vyzývajúce používateľa na zadanie adresy serveru.



Obr. 5.4 Dialógové okno zadávania adresy serveru

Ak používateľ zadá adresu a stlačí tlačidlo OK, pokúsi sa blok *Networking* vytvoriť spojenie s týmto serverom, a ak uspeje je poslaná požiadavka bloku *ChildWnd* na vytvorenie nového okna, ktoré bude prináležať práve vzniknutému spojeniu. Následne začne blok *Protocol* vykonávať prostredníctvom bloku *Networking* komunikačný protokol klientskej strany popísaný v kapitole 4.3.4.3.

Podobne server po prijatí spojenia a jeho asociácie s dokončovacím portom inicializuje vykonávanie serverového protokolu pre dané spojenie.

Prostredníctvom týchto dvoch protokolov je realizovaná výmena snímok medzi serverom a klientom. Vďaka použitiu *overlapped IO* modelu s *completion ports* je teoreticky možné aby tento server súčasne obsluhoval veľký počet klientov, keďže zvolený IO model veľmi efektívny.

Spojenie medzi serverom a klientom je ukončené v okamihu ak buď server ukončí svoju činnosť, ale sa klient rozhodne odpojiť.

5.1 Meranie reálnych parametrov aplikácie

Doteraz boli používané teoretické hodnoty parametrov prenosových médií na posudzovanie odhadovaných parametrov aplikácie. Keďže teraz je dostupná reálne funkčná aplikácie je možné zmerať pomocou tejto aplikácie niektoré parametre prenosového média.

5.1.1 Meranie rýchlosti prenosu

Podľa záverov v kapitole 3.1.3 by bol najvhodnejší prenosový prostriedok sieť *WiFi*, teda sieť postavená na štandarde IEEE 802.11. Keďže pri testovaní aplikácie bola dostupná len sieť IEEE 802.11g, bola použitá táto. Maximálna teoretická rýchlosť tejto bezdrôtovej siete je stanovená na 54 Mbit/s.

Pomocou vytvorenej aplikácie a nástroja operačného systému zvaného *Resource Monitor* bola zmeraná priemerná prenosová rýchlosť pri prenose na krátku vzdialenosť (do 5 metrov) približne 26,5 Mbit/s.

Vidno, že medzi teoretickou a reálnou rýchlosťou je veľmi výrazný rozdiel, tento rozdiel je zásadný najmä u bezdrôtových prenosových sietí, kde môže na prenášaný signál pôsobiť mnoho ručení a preto treba pri návrhu bezdrôtových prenosových systémov s týmto počítať.

5.1.2 Maximálna vzdialenosť

Zmeraná maximálna vzdialenosť, na ktorú bola schopná aplikácia komunikovať sa pohybuje v interiéry okolo 10 až 15 metrov, avšak pri krajných hodnotách vzdialeností klesla prenosová rýchlosť na veľmi malé, až takmer nepoužiteľné, hodnoty.

Vidno že aj tu teoretická hodnota 80 metrov je výrazne vyššia ako zmeraná reálna maximálna vzdialenosť.

Aby sa hodnoty maximálnej vzdialenosti priblížili teoretickej hodnote, bolo by nutné zabezpečiť priamu cestu signálu bez prekážok a minimalizovať možné rušenia odrazmi (*multipath distortion*).

5.1.3 Štatistika úspešnosti odoslaných dát

Keďže na prenos dát v programe bol použitý komunikačný protokol TCP/IP, ktorý zabezpečuje *reliabilitu* prenosu dát, nebolo možné zmerať štatistiku úspešnosti odoslaných dát.

6 ZÁVER

Cieľom práce bolo zvoliť vhodný bezdrôtový prenosový prostriedok schopný prenášať obraz medzi počítačmi a vytvoriť aplikáciu realizujúcu prenos obrazu pomocou tohto prenosového prostriedku.

Ako najvhodnejší prenosový prostriedok sa ukázala sieť *WiFi*, teda sieť založená na štandarde IEEE 802.11g. Táto rozhodne poskytuje dostatočnú rýchlosť na prenos plynulého komprimovaného videa. Avšak na surové obrazové dáta bez kompresie nemá dostatočný výkon.

Keďže z časových dôvodov sa nepodarilo implementovať do aplikácie kompresiu videa, nie je možné pomocou tohto prenosového prostriedku prenášať plynulý obraz v reálnom čase.

Vytvorená aplikácia používa veľmi efektívny spôsob práce s vstupne/výstupnými operáciami cez sieť nazývaný *overlapped IO with completion ports*. Toto zabezpečuje schopnosť aplikácie spracovávať obrovský počet pripojení naraz.

V priebehu realizácie bola aplikácia niekoľko krát prerábaná, keďže sa ukázalo, že niektoré prístupy nie sú vôbec vhodné na realizáciu takejto aplikácie. Ale aj niektoré postupy používané vo finálnej verzií aplikácie by bolo možné podstatne vylepšiť. Jednou z najdôležitejších zmien by bolo pridanie kompresie videa, keďže ako bolo od začiatku predpokladané s dostupnými prostriedkami nie je možné realizovať bezdrôtový prenos plynulého videa bez kompresie.

Pri meraní reálnych parametrov prenášaných dát bolo zistené že v mnohých ohľadoch sa zásadne líšia od predpokladaných teoretických hodnôt, čo je spôsobené neideálnym charakterom reálneho prenosového prostredia, s čím teoretické hodnoty zo zásady nepočítajú.

BIBLIOGRAFIA

[1] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thomson, 2008, 829 s. ISBN 978-049-5082-521.

[2] SHARP, John. *Microsoft Visual C# 2010*: *krok za krokem*. Vyd. 1. Brno: Computer Press, 2010, 696 s. ISBN 978-802-5131-473.

[3] BAYER, Jürgen. *C# 2005*: *velká kniha řešení*. Vyd. 1. Překlad Jiří Kolář. Brno: Computer Press, 2007, 813 s. ISBN 978-802-5116-203.

[4] PUŽMANOVÁ, Rita. *Moderní komunikační sítě od A do Z*. 2. aktualiz. vyd. Brno: Computer Press, 2006, 430 s. ISBN 80-251-1278-0.

[5] STALLINGS, William. *Wireless communications and networking*. Singapore: Pearson Education, 2002, 584 s. ISBN 81-780-8560-7.

[6] [online]. [cit. 2012-01-13]. Dostupné z: <http://www.yoda.arachsys.com/csharp/threads/>

[7] MICROSOFT. *MSDN Library* [online]. 2012 [cit. 2012-01-13]. Dostupné z: <http://msdn.microsoft.com/en-us/library/>

[8] ZIGBEE. *ZigBee Specifications* [online]. 2012 [cit. 2012-01-13]. Dostupné z: <http://www.zigbee.org/Specifications.aspx>

[9] VLČEK, Karel. *Komprese a kódová zabezpečení v multimediálních komunikacích*. 2. vyd. Praha: BEN - technická literatura, 2004, 258 s. ISBN 80-730-0134-9.

[10] Windows API. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-25]. Dostupné z: <http://en.wikipedia.org/wiki/Windows_API>

[11] JONES, Anthony a Amol DESHPANDE. Windows Sockets 2.0: Write Scalable Winsock Apps Using Completion Ports. *MSDN Magazine* [online]. 2000, October 2000 [cit. 2012-05-27]. Dostupné z: <http://msdn.microsoft.com/en-us/magazine/cc302334.aspx>

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

Príloha A: CD / DVD obsahujúce elektronickú verziu bakalárskej práce, zdrojové kódy aplikácie a preloženú aplikáciu