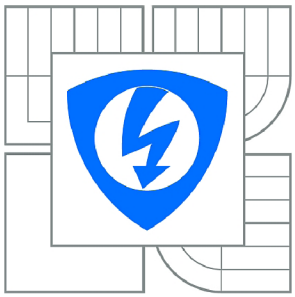
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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

DÁLKOVÉ SOFTWAROVÉ OVLÁDÁNÍ PLATFORMY BEZDRÁTOVÉHO MODULU MOBILNÍHO ROBOTU

REMOTE CONTROL OF REMOTE COMMUNICATION BOARD OF MOBILE ROBOT

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

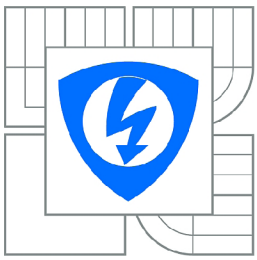
AUTOR PRÁCE — Bc. TOMÁŠ JÍLEK

AUTHOR

VEDOUCÍ PRÁCE — Ing. FRANTIŠEK BURIAN

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Tomáš Jílek	***ID:***	73006
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Dálkové softwarové ovládání platformy bezdrátového modulu mobilního robotu

POKYNY PRO VYPRACOVÁNÍ:

Na základě zjištěných možností dálkového softwarového řízení platformy RouterBoard od firmy Mikrotik realizujte aplikaci v jazyce C, která bude umožňovat z počítače dálkově řídit základní funkce platformy pomocí rodiny protokolů TCP/IP. Množinu ovládaných funkcí volte po dohodě s vedoucím práce pro použití v mobilní teleprezenční robotice.

DOPORUČENÁ LITERATURA:

MIKROTIK,: MikroTik wiki. <dostupné online: http://wiki.mikrotik.com>[cit. 29.1 2009]

Termín zadání:	7.2.2011	***Termín odevzdání:***	23.5.2011

Vedoucí práce: Ing. František Burian

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Práce se zabývá možnými způsoby vzdálené správy zařízení s operačním systémem MikroTik RouterOS a výběrem vhodné varianty pro realizaci automatické vzdálené správy tohoto zařízení, které je součástí mobilního robotu. Vzhledem k omezeným možnostem embedded systému, ze kterého je třeba provádět automatickou vzdálenou správu zařízení s tímto operačním systémem, nebylo možné použít standardizované protokoly určené pro tento účel. Podstatná část této práce proto řeší analýzu a detailní dokumentaci proprietárních protokolů implementovaných v operačním systému MikroTik RouterOS, které jsou vhodné pro implementaci do embedded systému, ze kterého je třeba provádět automatickou vzdálenou správu. V závěru práce je popsán princip navržené a realizované implementace proprietárních protokolů v jazyce C. Popisovaná implementace je součástí knihovny, která umožňuje automatickou vzdálenou správu s využitím těchto protokolů.

Klíčová slova

Embedded systém, MikroTik, RouterOS, vzdálená správa

Abstract

The thesis discusses the possible ways of remote management of devices running Mikrotik RouterOS and choosing appropriate variant for implementation of automated remote management of the device, which is part of a mobile robot. Due to the limited capabilities of embedded system, from which it is necessary to perform automated remote management of device with this operating system, it was not possible to use standardized protocols for this purpose. A substantial part of this work therefore deals with analysis and detailed documentation of proprietary protocols implemented in the MikroTik RouterOS operating system, which are suitable for implementation in an embedded system, from which it is necessary to perform automated remote management. The conclusion describes the principle of the proposed and realized implementation of proprietary protocols in C language. The described implementation is part of the library, which provides automated remote management using these protocols.

Keywords

Embedded system, MikroTik, RouterOS, remote management

Bibliografická citace:

JÍLEK, T. *Dálkové softwarové ovládání platformy bezdrátového modulu mobilního robotu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 77 s. Vedoucí diplomové práce byl Ing. František Burian.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Dálkové softwarové ovládání platformy bezdrátového modulu mobilního robotu 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 části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.“

V Brně dne: **23. května 2011** .....

podpis autora

Poděkování

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

V Brně dne: **23. května 2011**

.....
podpis autora

Obsah

1 ÚVOD

Jedna z mobilních platforem, která je vyvíjena na ústavu automatizace a měřicí techniky, využívá pro bezdrátový přenos dat mezi touto platformou a okolím platformu s operačním systémem MikroTik RouterOS. Tento operační systém je v dané aplikaci provozován na embedded zařízeních MikroTik RouterBOARD, které jsou primárně předurčeny pro provoz s tímto operačním systémem. Po korektním spuštění tohoto operačního systému v embedded zařízení je možné toto zařízení a operační systém lokálně i vzdáleně spravovat. Pro vzdálenou správu je možné využít jednak běžně dostupné aplikace využívající podporované standardizované protokoly (SSH, HTTP, ...), ale i specializované aplikace dodané výrobcem, které jsou výhradně určeny pro tento účel. Výrobce operačního systému distribuuje uvedený operační systém jako uzavřený, a z tohoto důvodu je tedy rozsah v něm implementovaných funkcí závislý pouze na výrobci tohoto operačního systému. Jakékoliv vlastní rozšíření na úrovni operačního systému nebo jednotlivých aplikací není možné.

Během praktického provozu mobilní platformy vyvstal požadavek na možnost automatické správy embedded zařízení zajišťujícího bezdrátový přenos dat. Novější verze operačního systému MikroTik RouterOS již disponují jednoduchým API, které je určeno právě pro využití v souvislosti s automatickou vzdálenou správou zařízení s tímto operačním systémem. Vlastnosti toho API jsou ovšem pro použití v dané konkrétní aplikaci nevhodné. Všechny ostatní způsoby vzdálené správy jsou předurčeny spíše pro ruční způsob vzdálené správy.

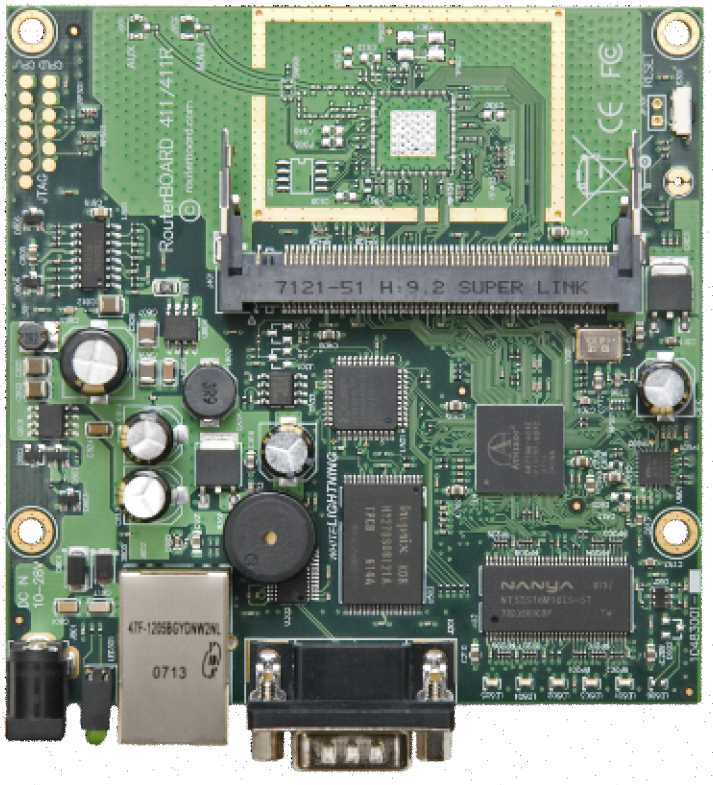
Cílem popisovaného projektu je zjistit dostupné způsoby správy zařízení, v nichž je korektně spuštěn operační systém MikroTik RouterOS a možnosti těchto jednotlivých zjištěných variant. Z této množiny variant je třeba zvolit jednu, která bude mít nejvhodnější vlastnosti pro použití v automatické vzdálené správě zařízení s operačním systémem MikroTik RouterOS. Jedním z požadavků je, aby bylo možné konfigurovat co největší množinu parametrů zařízení s tímto operačním systémem. Nejlépe, aby tato množina nastavení odpovídala množině nastavení, která je v tomto operačním systému možná. Klíčová je také implementovatelnost zvoleného řešení do embedded systému mobilní platformy a systémové nároky tohoto řešení.

Výchozím problémem ovšem je, že žádný z běžných, a tedy i zdokumentovaných protokolů, které operační systém MikroTik RouterOS pro svoji správu podporuje, nesplňuje požadavky na jeho potřebné vlastnosti. Jednou z možností je proto pokusit se analyzovat proprietární protokoly výrobce operačního systému MikroTik RouterOS, které využívá ve svých speciálních aplikacích určených pro správu. V případě úspěšné analýzy těchto protokolů a splnění podmínek na jejich vlastnosti je třeba vytvořit detailní popis jedné ze sestav těchto protokolů tak, aby ve výsledku bylo možné vytvořit knihovnu funkcí v jazyce C, která bude umožňovat použití zvolené sestavy protokolů pro automatickou vzdálenou správu zařízení s OS MikroTik RouterOS.

2 ZAŘÍZENÍ S OS MIKROTIK ROUTEROS

Operační systém MikroTik RouterOS může být obecně provozován na platformách mips, PowerPC a x86. Samotný operační systém je založen na Linuxu, ale výrobce nezveřejňuje jeho zdrojové kódy a systém distribuuje jako uzavřený s vlastní strukturou rozšiřujících balíčků. Nejčastější použití tohoto síťového operačního systému je na specializovaném HW MikroTik RouterBOARD. Jedná se o malé základní desky od firmy MikroTik obsahující procesor, paměť RAM, paměť FLASH, ethernetové porty, port sériového rozhraní RS-232 a často také rozšiřující sloty sběrnice miniPCI. Některé varianty jsou také vybaveny rozhraním USB, slotem pro rozšiřující paměťovou kartu, případně mají přímo na základní desce integrovánu bezdrátovou kartu. Obecně lze tyto základní desky použít jako aktivní síťová zařízení s volitelnou funkcí (bezdrátový přístupový bod, most, směrovač, klient, ...).

V uvažované mobilní platformě, která tvoří základ robotu ORPHEUS jsou nejčastěji použity jednotky MikroTik RouterBOARD RB411AH s rozšiřující bezdrátovou kartou se sběrnicí miniPCI pro zvolené frekvenční pásmo. Fotografie zmiňované desky je na následujícím obrázku:



Obrázek 2.1: MikroTik RouterBOARD RB411AH [1]

3 ZPŮSOBY SPRÁVY ZAŘÍZENÍ S OS MIKROTIK ROUTEROS

Pokud je zařízení s OS MikroTik RouterOS, na kterém je třeba provádět správu, vybaveno sériovým rozhraním RS-232, lze toto rozhraní konfigurovat jako port lokální konzole systému a pomocí této konzole zařízení lokálně spravovat. Správa zařízení je ovšem možná až po korektním spuštění OS MikroTik RouterOS.

Na zařízení s korektně spuštěným OS MikroTik RouterOS je možné vzdáleně provádět správu prostřednictvím komunikační sítě, ke které je toto zařízení připojeno. Pro vzdálenou správu zařízení s OS MikroTik RouterOS lze použít aplikační protokoly uvedené v následující tabulce:

Aplikační protokol	Transportní protokol	Výchozí porty	Sestava protokolů
FTP	TCP	20,21,> 1023	TCP-FTP
SSH	TCP	22	TCP-SSH
Telnet	TCP	23	TCP-Telnet
HTTP	TCP	80	TCP-HTTP
HTTPS	TCP	443	TCP-TLS-HTTP
SNMP	UDP	161	UDP-SNMP
MT	UDP	20561	UDP-MTP-MT
MWB	TCP	8291	TCP-MWB
MWB	UDP	20561	UDP-MTP-MWB
MWBS	TCP	8291	TCP-TLS*-MWB
MWBS	UDP	20561	UDP-MTP-TLS*-MWB
MAPI	TCP	8728	TCP-MAPI

Tabulka 3.1: Seznam podporovaných protokolů pro vzdálenou správu

* přítomnost tohoto protokolu nebyla experimentálně potvrzena

Ke vzdálené správě pomocí standardizovaných protokolů (FTP, SSH, Telnet, HTTP, HTTPS, SNMP) je nutné využít aplikací třetích stran, což nepředstavuje problém, protože se jedná o zcela běžné komunikační protokoly.

Protokol FTP (File Transfer Protocol) je nezabezpečený komunikační protokol pro přenos souborů mezi dvěma účastníky pomocí komunikační sítě. V úloze správy

zařízení s OS MikroTik RouterOS jej lze použít zejména pro uložení nebo stažení konfiguračních skriptů do/z tohoto zařízení. Případně lze tímto způsobem do zařízení nahrát softwarové balíčky rozšiřující funkce celého systému (soubory *.npk) nebo stáhnout soubory uložené na diskovém prostoru tohoto zařízení (např. obsah zaznamenaných rámců).

Protokol SSH (Secure Shell) je zabezpečený komunikační protokol primárně určený pro zabezpečené připojení ke vzdálenému systému přes nedůvěryhodnou síť. V souvislosti s OS MikroTik RouterOS jej lze využít ke zprostředkování zabezpečeného přístupu k příkazové řádce tohoto systému.

Protokol Telnet je nezabezpečený komunikační protokol zprostředkovávající přístup k příkazové řádce vzdáleného systému. Princip práce se vzdáleným systémem je stejný, jako při použití protokolu SSH. Nejzásadnějším problémem protokolu Telnet ale v dnešní době je, že samotný protokol neobsahuje jakékoliv bezpečnostní prvky (šifrování hesel a samotných dat, kontrolu integrity přijatých dat, ...).

Protokol HTTP (Hypertext Transfer Protocol) je nezabezpečený komunikační protokol určený pro přenos hypertextových dokumentů ve formátu HTML. Ve spojitosti s OS MikroTik RouterOS lze tento protokol využít pro přístup k webové službě WebBox (resp. WebFig v novějších verzích), která umožňuje vzdálenou správu systému pomocí grafického rozhraní. To může být výhodou oproti textovému přístupu ke vzdálené správě systému.

Protokol HTTPS (Hypertext Transfer Protocol Secure) je zabezpečenou variantou protokolu HTTP pomocí protokolu TLS (Transport Layer Security). Přenášená data jsou tak chráněna proti odposlechu a podvržení a současně je umožněno ověření identity protistrany.

Protokol SNMP (Simple Network Management Protocol) je speciální protokol pro správu síťových prvků jako jsou např. přepínače, routery a jiná aktivní síťová zařízení. Tento protokol lze využít pro monitoring celé sítě, ve které je obsaženo velké množství síťových zařízení. Protokol samotný umožňuje čtení a zápis hodnot proměnných z/do síťového zařízení. Protokol je tedy uzpůsoben spíše pro monitoring vybraných parametrů síťového zařízení, než na jeho komplexní správu. Největším nedostatkem implementace v OS MikroTik RouterOS je velmi omezená množina operací, které vyžadují zápis dat do zařízení. Tato množina je omezena pouze na změnu identity zařízení, restartování OS MikroTik RouterOS nebo spuštění vybraného skriptu s jedním parametrem. Praktické použití tohoto protokolu má tedy smysl jen pro monitoring zařízení s OS MikroTik RouteOS a nikoliv pro jeho různorodou parametrizaci.

Protokol MT (MikroTik Terminal) představuje proprietární komunikační protokol zprostředkovávající přístup k příkazové řádce vzdáleného systému. Princip práce se vzdáleným systémem je stejný, jako při použití protokolu SSH nebo Telnet. Přenos hesla je zabezpečen použitím hašovací funkce MD5. Ostatní data ale nejsou nijak zabezpečena. Podstatným rozdílem oproti použití protokolů SSH nebo Telnet je ten, že

tento protokol využívá transportní protokol UDP. Protokoly SSH nebo Telnet využívají transportní protokol TCP. Další zásadní odlišností je způsob identifikování cílového zařízení. Protokoly SSH a Telnet využívají v konečném důsledku IP adresu cílového zařízení. Protokol MT pracuje pouze s MAC adresou síťového rozhraní cílového zařízení. IP adresu není třeba znát, ani není třeba, aby ji měli oba účastníci přidělenu. Protokol MT využívá broadcastové vysílání pomocí UDP protokolu. Z toho ovšem plyne omezení, že spojení mezi dvěma účastníky je možné jen v rámci jednoho segmentu sítě. Důvodem je použití směrovačů mezi jednotlivými segmenty sítě, které tento typ komunikace do sousedních segmentů sítě nepropouštějí. Pro použití uvedeného protokolu je třeba využít výrobcem dodávanou aplikaci Terminal nebo druhé zařízení s OS MikroTik RouterOS, které bude ve funkci klienta. Připojení ke klientskému zařízení je možné provést libovolným způsobem, který umožňuje zadávat příkazy konzole (lokální připojení přes sériové rozhraní RS-232, MWB/MWBS, MT, SSH, Telnet).

Protokol MWB (MikroTik WinBox) představuje proprietární komunikační protokol umožňující realizaci libovolné operace v zařízení s OS MikroTik RouterOS. Pokyny k operacím a výsledky těchto operací jsou ale přenášeny v binárních datech bez jakékoliv jejich textové interpretace. Z tohoto způsobu komunikace plyne, že při rozšíření funkcí na straně serveru je nutné aktualizovat i aplikaci na straně klienta. V případě správy systému pomocí textové konzole toto ale nutné není. Klientská aplikace zůstane stejná. Uvedený komunikační protokol je využíván výrobcem dodávanou aplikací WinBox.

Protokol MWBS (MikroTik WinBox Secure) představuje zabezpečenou variantu protokolu MWB. Úroveň zabezpečení tohoto protokolu není známa, lze ale předpokládat minimálně ochranu proti odposlechu.

Protokol MikroTik API (MAPI) představuje proprietární, ale zdokumentovaný protokol pro vzdálenou správu OS MikroTik RouterOS. Pro jeho použití není výrobcem dodávána žádná aplikace. Výrobce pouze zveřejnil princip fungování tohoto protokolu a několik vzorových zdrojových kódů v nejběžnějších programovacích jazycích (C, C++, php, Python, ...), ve kterých se ale vyskytují závažné programátorské chyby mající vliv na výslednou funkčnost. Tento komunikační protokol je určen pro vzdálenou správu zařízení s OS MikroTik RouterOS z uživatelských zařízení, do kterých si jej uživatelé mohou implementovat sami. Nevýhodou je ovšem jeho neustálená podoba v různých verzích OS MikroTik RouterOS, což vede k problémům, kdy po instalaci nové verze OS MikroTik RouterOS do spravovaného zařízení, následná komunikace nefunguje správně.

4 APLIKACE PRO SPRÁVU ZAŘÍZENÍ S OS MIKROTIK ROUTEROS

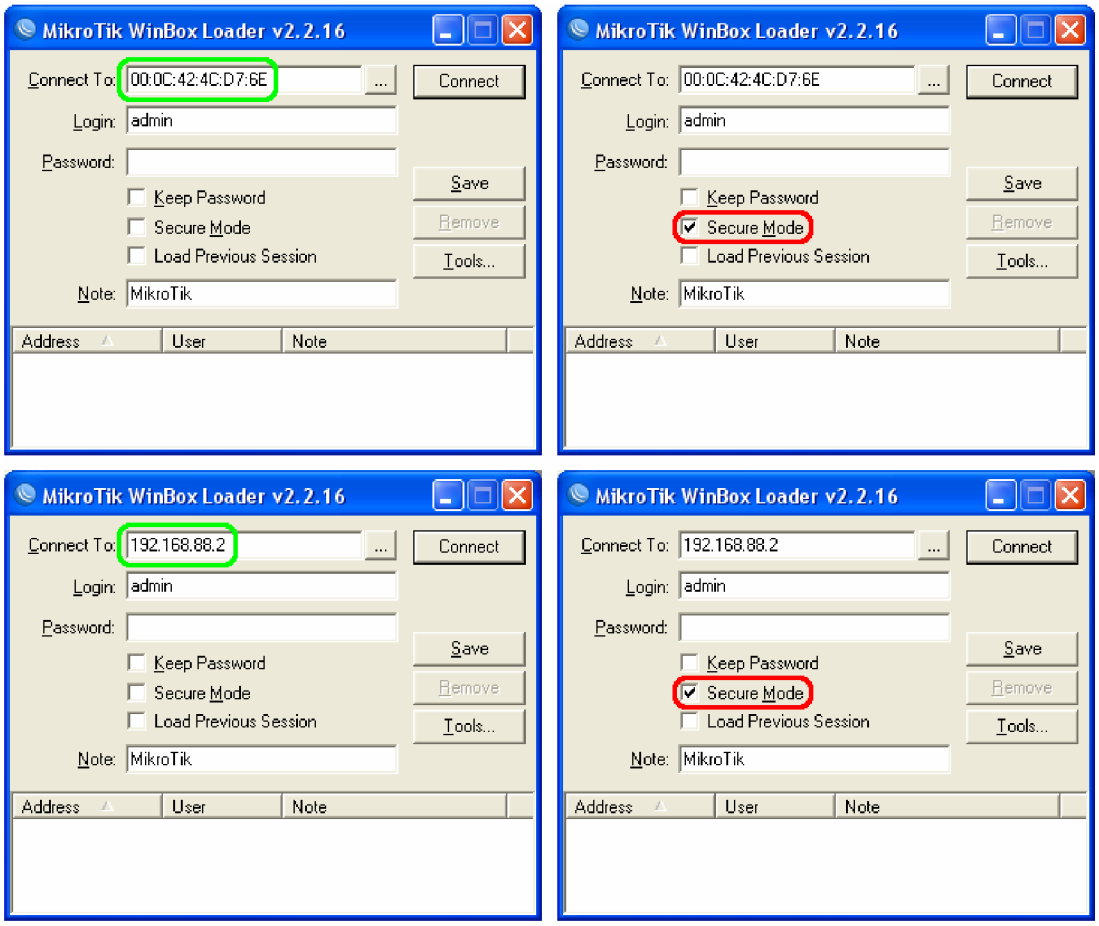
Výrobce OS MikroTik RouterOS dodává k jeho správě dvě základní aplikace. První aplikací je aplikace WinBox a druhou aplikací je aplikace MikroTik Neighbor Viewer, jejíž součástí je aplikace Terminal. Ani jedna z aplikací nevyužívá standardizovaných komunikačních protokolů na aplikační vrstvě komunikačního modelu, ale využívá vlastní proprietární protokoly s veřejně nepřístupnou dokumentací.

Pro využití standardizovaných protokolů na aplikační vrstvě komunikačního modelu je možné použít libovolné aplikace pracující s daným protokolem podporovaným OS MikroTik RouterOS pro jeho správu. Jedná se zejména o využití protokolů SSH, Telnet, HTTP, HTTPS a SNMP.

4.1 Aplikace WinBox

Aplikace WinBox je aplikace s grafickým uživatelským rozhraním, určená ke správě zařízení s OS MikroTik RouterOS. Aplikace je určena pouze pro běh pod operačními systémy Microsoft Windows. Samotný spustitelný soubor winbox.exe představuje pouze zavaděč dynamicky linkovaných knihoven, které realizují vlastní konfigurační funkce v aplikaci WinBox. Obsah souboru winbox.exe i obsah přidružených dynamicky linkovaných knihoven se velmi často mění (prakticky s každou novou verzí OS MikroTik RouterOS). Z toho je patrné, že aplikace WinBox je aktivně vyvíjenou aplikací, z čehož plyne i riziko, že komunikační protokol použitý v různých verzích OS MikroTik RouterOS může být nekompatibilní s verzí použitou v jiné verzi OS MikroTik RouterOS. Zavaděč konfiguračních dynamicky linkovaných knihovem (soubor winbox.exe) lze stáhnout přímo ze zařízení na kterém je korektně spuštěn OS MikroTik RouterOS, např. pomocí webového prohlížeče protokolem HTTP z adresy `http://ip_address/winbox/winbox.exe`, kde *ip_address* označuje IP adresu zařízení, na které běží webový server konfigurační služby.

Samotná komunikace mezi aplikací WinBox a zařízením s OS MikroTik RouterOS může probíhat celkem ve čtyřech různých režimech. První volbou je, zda je pro připojení zadána MAC adresa nebo na IP adresa. Druhým nastavením je, zda-li je požadováno zabezpečené spojení či nikoliv. Zabezpečený režim komunikace se volí vybráním volby *Secure Mode*. Příklady všech čtyřech možných kombinací jsou uvedeny na obrázku na následující straně. Příklad je uveden pro WinBox Loader ve verzi 2.2.16, který je obsažen v OS MikroTik RouterOS ve verzi 4.11.



Obrázek 4.1: Možnosti nastavení komunikace v aplikaci WinBox

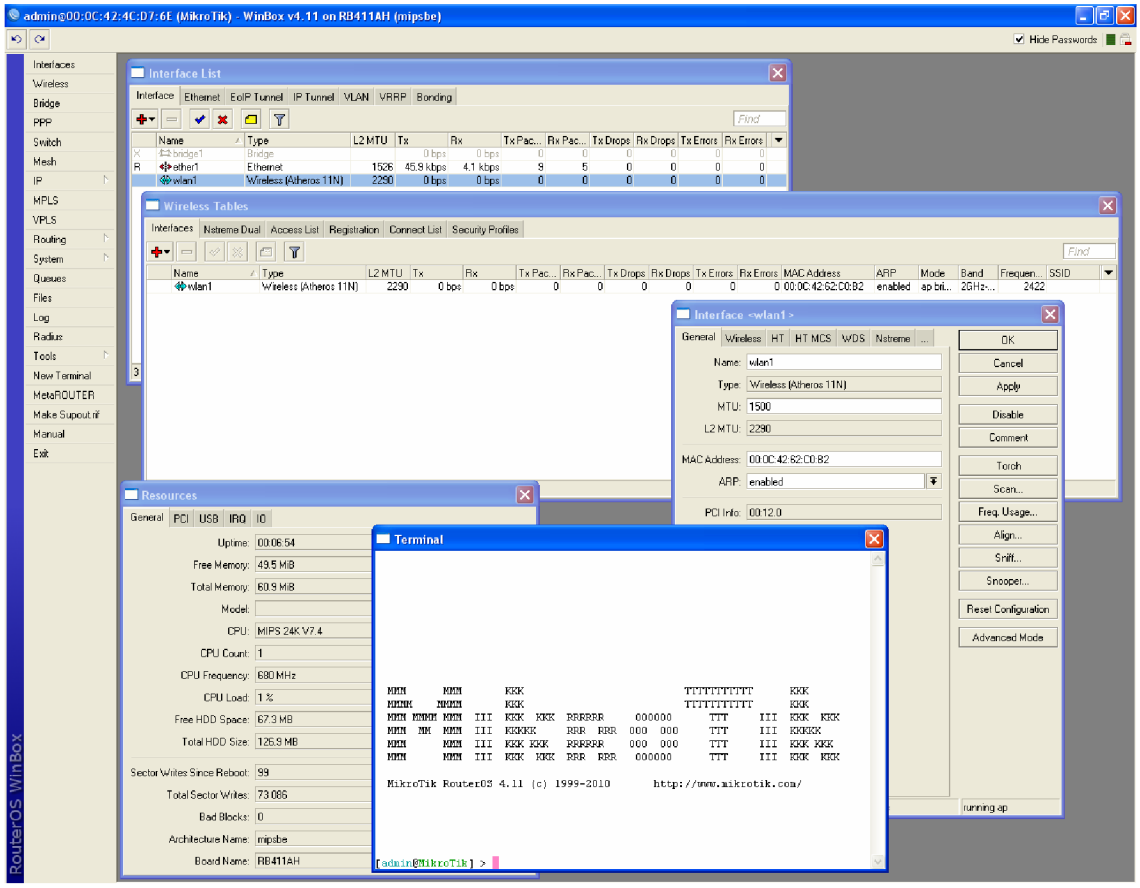
V případě, kdy je zadána MAC adresa, je na transportní vrstvě použit protokol UDP a na aplikační vrstvě protokol MTP, který nese v případě nezabezpečeného spojení data protokolu MWB a v případě zabezpečeného spojení data protokolu MWBS. V situaci, kdy je zadána IP adresa, je ke komunikaci na transportní vrstvě použit protokol TCP a na aplikační vrstvě již přímo protokol MWB v případě nezabezpečeného spojení nebo protokol MWBS v případě zabezpečeného spojení. Popisované použité sestavy protokolů jsou uvedeny v následující tabulce:

		Režim komunikace	
		Nezabezpečený	Zabezpečený
Typ zadané adresy	MAC	UDP-MTP-MWB	UDP-MTP-MWBS
	IP	TCP-MWB	TCP-MWBS

Tabulka 4.1: Použité sestavy protokolů pro různé režimy komunikace

Po stisku tlačítka *Connect* si aplikace vyžádá ze zařízení s OS MikroTik RouterOS informace o verzích dynamicky linkovaných knihoven uložených v tomto zařízení. Tyto

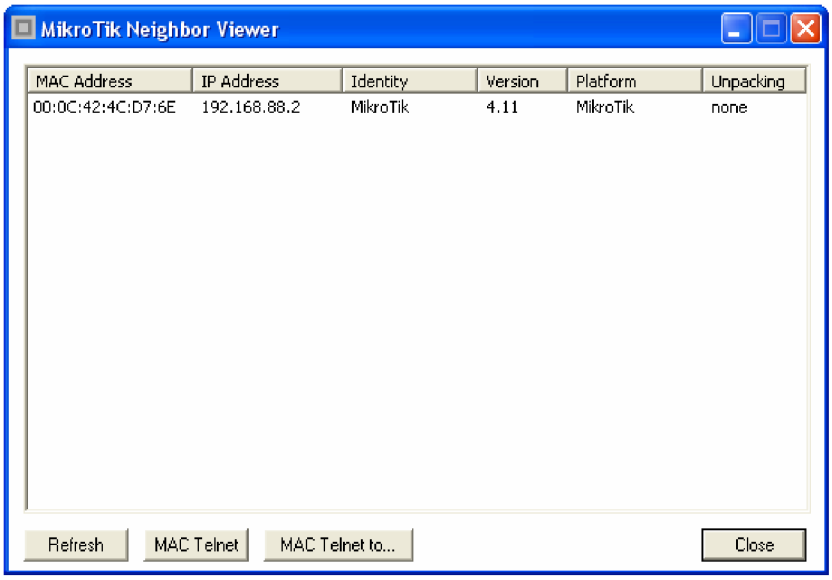
získané informace o verzích srovná s informacemi o verzích lokálně uložených knihoven (pokud již byly někdy staženy). V případě, kdy jsou na cílovém zařízení knihovny novější, stáhne a následně použije tyto nové verze dynamicky linkovaných knihoven. V případě, kdy je zvolen zabezpečený režim komunikace, není tato úvodní část komunikace ještě šifrována a je tedy stejná jak pro nezabezpečený tak i pro zabezpečený režim komunikace. Ukázka okna aplikace WinBox je na následujícím obrázku (MikroTik RouterOS 4.11, WinBox Loader 2.2.16):



Obrázek 4.2: Okno aplikace WinBox

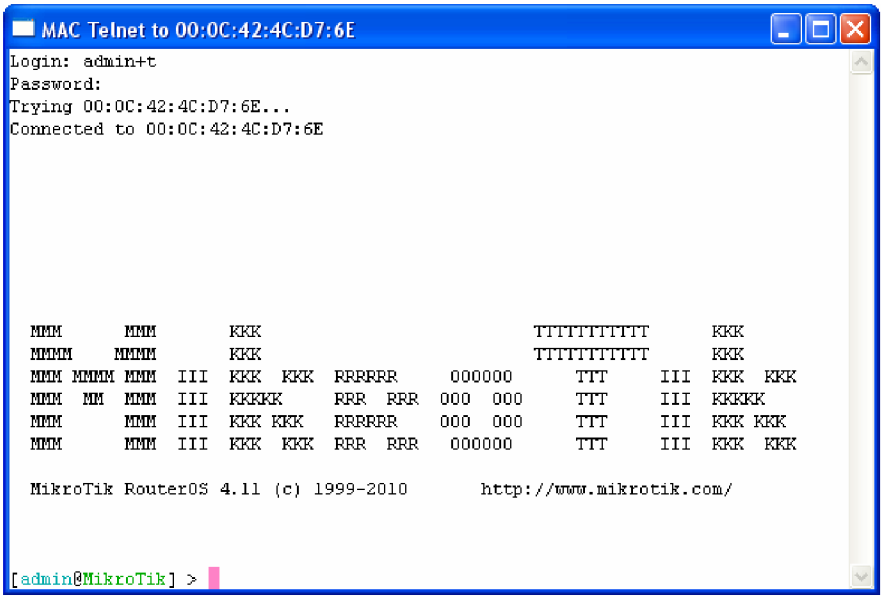
4.2 Aplikace Terminal

Aplikace Terminal je součástí aplikace MikroTik Neighbor Viewer. Aplikace MikroTik Neighbor Viewer slouží k vyhledání zařízení na síti pomocí Cisco Discovery protokolu (CDP) a pomocí MikroTik Neighbor Discovery protokolu (MNDP). Ukázka okna této aplikace je na obrázku na následující straně.



Obrázek 4.3: Okno aplikace MikroTik Neighbor Viewer

Aplikace Terminal zajišťuje připojení ke vzdálenému zařízení s OS MikroTik RouterOS pomocí komunikační sítě (např. IEEE 802.3 Ethernet), ke které je PC s touto aplikací a spravované zařízení připojeno. K připojení je nutná znalost pouze MAC adresy síťového rozhraní cílového zařízení, na kterém naslouchá server příslušné služby. IP adresu nemusí mít cílové zařízení přidělenou. MAC adresa síťového rozhraní cílového zařízení, ke kterému je požadováno se připojit, se zadává jako argument aplikace Terminal, tedy např. `terminal.exe 00:0C:42:4C:D7:6E`. Ukázka okna této aplikace je na následujícím obrázku:



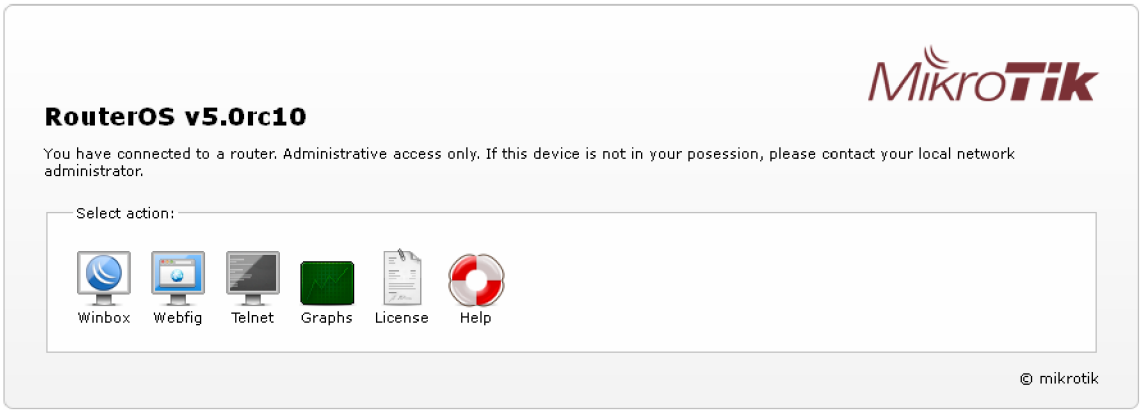
Obrázek 4.4: Okno aplikace Terminal

Vzhledem k tomu, že tato aplikace nemá žádné volby režimu svého běhu, je způsob komunikace pouze jeden. Na transportní vrstvě je použit protokol UDP a na aplikační vrstvě je použit protokol MTP, který nese data protokolu MT.

Datum vytvoření souboru terminal.exe v zip archivu, který lze stáhnout z webu výrobce operačního systému, je 7.9.2004. Jiná verze souboru dostupná není. Z toho je patrné, že aplikace v posledních letech nedoznala změny. Na základě tohoto faktu lze usuzovat, že použitá sestava protokolů MTP-MT v novějších verzích OS MikroTik RouterOS by měla být zpětně kompatibilní se sestavou protokolů MTP-MT implementovanou v jeho starších verzích. Nemělo by tedy hrozit riziko nekompatibility, které hrozí u sestav protokolů využívajících protokol MWB nebo MWBS.

4.3 Webový prohlížeč

Pomocí webového prohlížeče lze přistoupit ke službě WebBox, která běží na webovém serveru spuštěného v zařízení s OS MikroTik RouterOS. Tuto službu je možné využít pro správu zařízení s OS MikroTik RouterOS. Rozsah implementovaných funkcí je však např. oproti aplikaci WinBox značně omezen. Od verze 5 OS MikroTik RouterOS byla webová služba WebBox přejmenována na WebFig a rozsah implementovaných funkcí byl značně rozšířen, téměř k rozsahu aplikace WinBox. Do webového prohlížeče stačí vložit pouze IP adresu, na které naslouchá webový server s touto službou. Ve výchozí konfiguraci naslouchá tento server pro nezabezpečené připojení na standardním portu TCP/80 (HTTP) a pro zabezpečené připojení na standardním portu TCP/443 (HTTPS). Ukázka úvodních webových stránek služeb WebFig a WebBox, které jsou zobrazovány ve webovém prohlížeči, je na následujících obrázcích:



Obrázek 4.5: Úvodní webová stránka služby WebFig (MikroTik RouterOS 5.0rc10)

Obrázek 4.6: Úvodní webová stránka služby WebBox (MikroTik RouterOS 4.11)

4.4 SSH klient

SSH klientem je možné se připojit k zařízení s OS MikroTik RouterOS zabezpečeným protokolem SSH prostřednictvím komunikační sítě, ke které je spravované zařízení připojeno. Správa zařízení se prostřednictvím tohoto protokolu provádí pomocí textové konzole, jejíž interpreter je společný pro všechny protokoly, které umožňují zprostředkovat příkazovou řádku vzdáleného systému (SSH, Telnet, MT). Stejný interpreter se využívá i při lokálním připojení k zařízení přes sériové rozhraní RS-232. Jako SSH klienta lze použít např. aplikaci PuTTY. Ukázka okna této aplikace je na obrázku na následující straně.

Obrázek 4.7: Okno aplikace PuTTY

4.5 Telnet klient

Pomocí klienta pro protokol Telnet je možné se k zařízení připojit nezabezpečeným protokolem Telnet prostřednictvím komunikační sítě, ke které je spravované zařízení připojeno. Samotná správa zařízení probíhá opět pomocí textové konzole. Jako klienta pro protokol Telnet lze využít aplikací PuTTY, HyperTerminál z operačního systému Microsoft Windows a jiných.

4.6 Terminál sériové komunikace

Pro správu zařízení lze také využít libovolnou terminálovou aplikaci umožňující komunikovat prostřednictvím sériového rozhraní RS-232. Pro správu zařízení je opět použita textová konzole. Jako terminálovou aplikaci lze opět použít např. aplikace PuTTY, HyperTerminál z operačního systému Microsoft Windows a jiné.

5 VÝBĚR VHODNÉHO ŘEŠENÍ

V zadání je požadována implementovatelnost řešení automatické vzdálené správy zařízení s OS MikroTik RouterOS do embedded systému použitého pro řízení mobilní platformy. Uvažovaný embedded systém, ze kterého má být prováděna automatická vzdálená správa, nemá v současné době na transportní vrstvě komunikačního modelu implementován protokol TCP, ale jen protokol UDP. Z uvedeného zásadního omezení je patrné, že dále má smysl uvažovat z aplikačních protokolů implementovaných v OS MikroTik RouterOS pouze protokoly SNMP, MT, MWB a MWBS, které pracují nad protokolem UDP. Protokol SNMP lze pro daný účel použití také vyřadit, protože umožňuje na cílovém systému přímo realizovat jen velmi omezenou množinu operací (změna identity, restart, spuštění skriptu). Nepřímo lze tuto množinu rozšířit použitím předem připraveného skriptu, který lze vzdáleně tímto protokolem spustit. Skript může provést předem vybranou operaci s využitím jednoho předaného parametru. Tento způsob automatické vzdálené správy je ale neakceptovatelný, protože by vyžadoval udržovat na všech spravovaných zařízeních množinu potřebných skriptů.

Z protokolů implementovaných v OS MikroTik RouterOS pro jeho správu zbývá využití protokolů MT, MWB/MWBS. Zásadním problémem ovšem je, že protokoly MTP, MT a MWB/MWBS jsou proprietárními protokoly výrobce operačního systému a výrobce k nim odmítá vydat jejich dokumentaci. Jedinou možností je tedy pokusit se analyzovat průběhy komunikací využívajících sestav protokolů MTP-MT, MTP-MWB nebo MTP-MWBS.

Další možnou cestou, jak vzniklý problém řešit, je upravit koncepci celého přenosu konfiguračních dat a dodat další člen do přenosové cesty. Vzhledem k tomu, že v projektu použitý hardware (MikroTik RouterBOARD) disponuje sériovým rozhraním RS-232, lze toto rozhraní použít jako port lokální konzole systému. Tak lze realizovat další způsob správy systému. Nevýhodou ovšem je pouze lokální přístup, což je pro daný účel neakceptovatelné řešení. Možným alternativním řešením je doplnit systém o převodník sériového rozhraní RS-232 na rozhraní typu IEEE 802.3 Ethernet, který by přijaté znaky na určitém UDP portu přeposílal na sériové rozhraní RS-232 a současně by znaky přijaté z rozhraní RS-232 odesílal na zvolený UDP port přes rozhraní typu IEEE 802.3 Ethernet. Využití sestavy protokolů MTP-MT má však jednu zajímanou výhodu oproti právě popisovanému řešení. Při použití protokolu MT není vyžadována znalost IP adresy spravovaného zařízení s OS MikroTik RouterOS, ale pracuje se vždy s broadcast IP adresou 255.255.255.255. Identifikace zařízení s OS MikroTik RouterOS tak probíhá až na úrovni MTP protokolu a je založena na znalosti MAC adresy jeho síťového rozhraní, na kterém běží server konfigurační služby. To je výhodné, neboť zařízení tak lze konfigurovat i v době, kdy např. ještě nemá korektně přidělené IP adresy nebo přidělené IP adresy nejsou známé. Žádný z běžně dostupných komerčních převodníků tuto zmíněnou funkcionalitu nenabízí. Navíc je nutné přihlédnout

k navýšení celkové elektrické spotřeby zařízení s OS MikroTik RouterOS vlivem použitého převodníku, protože uvedený převodník by byl použit v mobilních částech mobilní platformy. Při použití více zařízení s OS MikroTik RouterOS by bylo nutné vybavit všechna tato zařízení zmiňovaným převodníkem. Použití převodníku také ale může vést k dalším potencionálním problémům, jejichž zdroji může být právě popisovaný převodník. Nároky na vývoj vhodného převodníku jsou srovnatelné s nároky řešení využívající proprietární sestavu protokolů MTP-MT.

Druhé řešení vychází z předchozího, ale překlad aplikačních protokolů realizuje v jiném bodě přenosové trasy. Popisovaný překlad aplikačních protokolů nemusí probíhat až v místě konfigurovaného zařízení, kdy toto řešení vyžaduje u každého konfigurovaného zařízení s OS MikroTik RouterOS jeden převodník. Je možné využít jeden převodník v každém místě, kde vznikají požadavky konfigurace. Tj. např. v řídicím systému mobilního robotu. Popisovaný převodník by v této situaci nebylo nutné vyvíjet, protože by bylo možné použít např. jednotku RouterBOARD vybavenou sériovým rozhraním RS-232 a jedním ethernetovým portem. Připojení jednotky RouterBOARD k řídicímu systému robotu by bylo realizováno pomocí sériového rozhraní RS-232, které by bylo konfigurováno jako port lokální konzole systému. Pomocí této konzole by řídicí systém robotu zajistil autorizaci k OS MikroTik RouterOS spuštěného na jednotce RouterBOARD. Po úspěšné autorizaci by bylo možné spustit příkaz konzole, který realizuje připojení k jinému zařízení s OS MikroTik RouterOS, např. pomocí sestavy protokolů MTP-MT nebo protokolu SSH. Nevýhodou tohoto řešení je opět potřeba většího počtu převodníků, které by pracovaly jako konfigurační klienti. V případě použití jednotek RouterBOARD by nebylo možné mít v jeden okamžik z jednoho konfiguračního místa otevřeno více jak jedno konfigurační spojení na jiné zařízení s OS MikroTik RouterOS. Což by bylo ve skutečnosti velmi nevýhodné, protože by nebylo možné např. v jednom okamžiku provádět operace na více konfigurovaných zařízeních současně (např. měření datových toků na jednotlivých portech každého z konfigurovaných zařízení). Nebylo by možné mít ani spuštěný jeden dotazovací příkaz (např. pro vyčítání stavu) a pomocí druhého příkazu měnit současně nastavení na stejném zařízení s OS MikroTik RouterOS. V případě použití vlastní aplikační brány by bylo nutné tyto požadavky na současné připojení několika zařízení zohlednit.

Jedním z posledních řešení je použití otevřeného operačního systému ve virtuálním stroji, který je součástí OS MikroTik RouterOS. OS MikroTik RouterOS nabízí tuto funkcionalitu pod označením MetaRouter. Vhodným operačním systémem může být např. v současnosti jeden z nejpopulárnějších operačních systémů pro malé směrovače a aktivní síťová zařízení, operační systém OpenWRT. Řešení by spočívalo ve vytvoření aplikace, která by zasílala data přijatá vlastním navrženým protokolem využívajícím UDP protokol, standardizovaným komunikačním protokolem (API rozhraní OS MikroTik RouterOS, Telnet, ...) do OS MikroTik RouterOS. Největší slabinou tohoto

řešení je v současnosti nespolehlivá implementace virtuálního stroje v OS MikroTik RouterOS. Na některých typech HW bohužel poměrně často dochází k zamrzání operačního systému ve virtuálním stroji, nebo k pádu a následnému restartu celého OS MikroTik RouterOS. Funkce MetaRouter posouvá celý tento systém dobrým směrem, ale bohužel jeho implementace je v současné době prozatím nedořešená. Proto je v současnosti nevhodné implementovat nějaké klíčové funkce zařízení s OS MikroTik RouterOS pomocí operačního systému spuštěného v jeho virtuálním stroji.

Řešení založené na použití nedokumentovaných proprietárních protokolů přímo v zařízení, ve kterém vznikají požadavky ke konfiguraci zařízení s OS MikroTik RouterOS se nakonec jeví ze všech možných variant jako nejvhodnějším řešením celého problému. Bez modifikace v současnosti použitého HW a zásahu do nastavení jednotek MikroTik RouterBOARD by bylo možné splnit všechny vlastnosti kladené na nově navrhovanou automatickou vzdálenou správu zařízení s OS MikroTik RouterOS.

6 METODIKA ANALÝZY

Analýzu komunikace využívající sestavy protokolů MTP-MT je možné provádět při komunikaci mezi PC s aplikací Terminal a zařízením, na němž běží OS MikroTik RouterOS. Druhou možností je analyzovat komunikaci mezi dvěma zařízeními na nichž běží OS MikroTik RouterOS. Pro připojení ke klientskému zařízení, ze kterého se bude provádět vzdálená správa, je možné využít terminálového klienta pro připojení pomocí sériového rozhraní RS-232, klienta pro protokol SSH nebo Telnet, aplikaci Terminal případně aplikaci WinBox. Podstatné je, aby zvolený způsob připojení ke klientskému zařízení umožňoval zadávat příkazy konzole. V aplikaci WinBox je navíc možné využít interní funkce MAC-Telnet. Analýzu komunikace založenou na protokolu MWB nebo MWBS je možné provádět pouze při komunikaci mezi PC s aplikací WinBox a zařízením s OS MikroTik RouterOS.

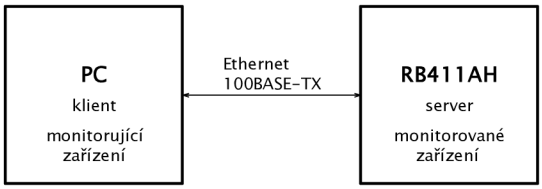
Vzhledem k povaze problému (nízké datové toky, analýza, nikoliv dlouhodobé sledování) lze bez problému použít běžné PC a nástroj Wireshark. Pro tento projekt je použita verze 1.4.0, která byla aktuální stabilní verzí v době zahájení analýzy použitých komunikačních protokolů (září 2010).

Z hlediska metodiky je nejprve odchyceno 5 shodných úspěšných přihlášení se ke vzdálenému systému zvoleným typem komunikace. V dalších krocích jsou měněny specifické parametry komunikace jako je MAC adresa klienta a serveru, uživatelské jméno, heslo, a další parametry a detekovány změny v zachycených ethernetových rámcích oproti výchozímu způsobu přihlášení se ke vzdálenému systému.

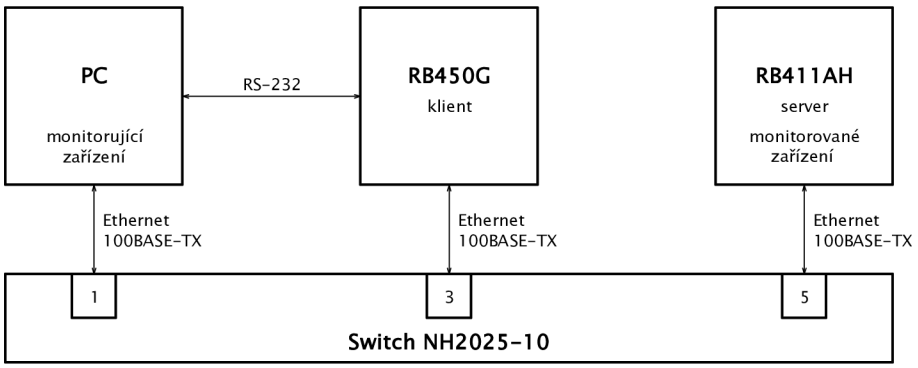
Zachycené průběhy komunikace jsou ukládány do souboru v otevřeném formátu *pcap* (Packet CAPture). Pro snadnou čitelnost a porovnávatelnost komunikace jsou ethernetové rámce komunikace exportovány do textové podoby, kdy v datové reprezentaci zůstávají pouze data nesená protokolem UDP. Nižší vrstvy jsou popsány pouze základními parametry uvedenými v jednom řádku pro každou vrstvu (viz. příloha A). Pro ještě snadnější porovnávání je výhodné mít k dispozici soubor obsahující pouze data nesená protokolem UDP. Tento úkol již však nástroj exportu ve Wiresharku ve standardní kompilaci neumí. Nejrychlejším a nejjednodušším způsobem je vyexportovaný soubor z Wiresharku vyfiltrovat na protokoly nižších vrstev než zkoumaných protokolů. To je možné provést jednoduchým klíčem, kdy jsou ponechány z exportovaného souboru pouze řádky začínající adresou nesených dat protokolem vyšším než je UDP. Tak lze velice efektivně porovnávat libovolné typy komunikace za použití libovolného inteligentního programu pro porovnávání obsahu souborů ve formátu prostého textu.

Při analyzování komunikace mezi PC s příslušnou aplikací a zařízením s OS MikroTik RouterOS stačí použít pouze Wireshark a monitorovat komunikaci na příslušném síťovém rozhraní PC. Při komunikaci mezi dvěma zařízeními s OS MikroTik RouterOS je zapotřebí zaslat také tuto komunikaci do příslušného rozhraní

PC. To je možné buď přímo za pomoci zařízení s OS MikroTik RouterOS nebo za použití opakovače (hubu) či managovatelného přepínače (switche). Vzhledem k tomu, že ze zařízení s OS MikroTik RouterOS je dostupný pouze MikroTik RouterBOARD RB450G, který má 5 portů a MikroTik RouterBOARD RB411AH, který má dostupný pouze 1 ethernetový port, by bylo vždy nutné použít MikroTik RouterBOARD RB450G jako síťový sniffer. Funkce RB450G by byla volena jako klient či server podle potřeby. Nevýhodou je, že by se role RB450G v některých případech měnila (klient nebo server), což by případně mohlo ovlivnit vlastnosti komunikace. Proto je komfortnější použít již zmiňovaný opakovač (hub) nebo managovatelný přepínač (switch). Opakovač (hub) s rozhraním IEEE 802.3 Ethernet ve variantě 100BASE-TX případně 10BASE-T nebyl dostupný, proto byl použit pro monitorování komunikace managovatelný přepínač NBase-Xyplex NH2025-10 s rozhraními IEEE 802.3 Ethernet ve variantě 100BASE-TX. Tento přepínač umožňuje zasílat komunikaci z vybraného fyzického portu na libovolný vybraný fyzický port. Toho je využito tak, že komunikace na fyzickém portu 5 je zasílána také do fyzického portu 1. Druhé zařízení se může nacházet v libovolném portu, např. 3. Oba používané případy propojení jsou zachyceny na následujících obrázcích:



Obrázek 6.1: Analýza komunikace mezi PC a zařízením



Obrázek 6.2: Analýza komunikace mezi dvěma zařízeními

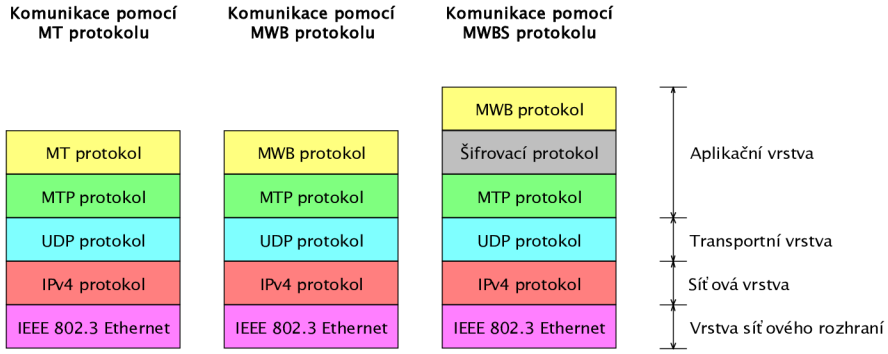
Analýza byla provedena na komunikacích, kdy jako server byla použita jednotka MikroTik RouterBOARD RB411AH s OS MikroTik RouterOS ve verzi 4.11. Ve vybraných případech byla jako klient použita jednotka MikroTik RouterBOARD RB450G s OS MikroTik RouterOS ve verzi 4.11.

7 TCP/IP MODEL KOMUNIKACE

Tato kapitola popisuje komunikaci využívající na aplikační vrstvě komunikačního modelu proprietární protokol MikroTik Transmission Protocol (MTP), který nese buď data protokolu MikroTik Terminal (MT) nebo data protokolu MikroTik WinBox (MWB) nebo data protokolu MikroTik WinBox Secure (MWBS). Dále je podrobně rozebrán proprietární protokol MT. Proprietární protokoly MWB a MWBS jsou zmíněny jen okrajově. Uvedený popis je vytvořen na základě provedených analýz několika typů komunikací využívajících zmíněné protokoly.

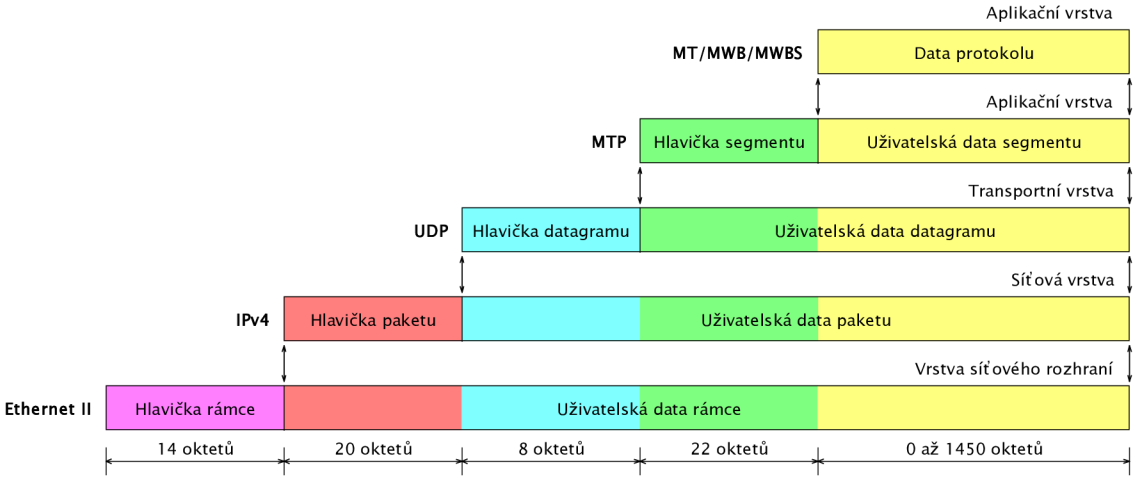
Z provedených analýz vyplynulo, že protokol v aplikaci Terminal, který nese data protokolu MT je prakticky téměř shodný s protokolem použitým v aplikaci WinBox při připojení na základě zadané MAC adresy, který nese data protokolu MWB v nezabezpečeném režimu komunikace, nebo data protokolu MWBS v zabezpečeném režimu komunikace. Význam i obsah jednotlivých oktetů v MTP segmentu je naprosto shodný. Rozdíly jsou až na úrovni chování protokolu jako celku (na úrovni stavového diagramu). Proto je na uvedený protokol použitý v aplikacích Terminal a WinBox dále pohlíženo jako na jeden protokol. Zda-li se opravdu jedná o jeden a ten samý komunikační protokol, ovšem jen s jiným nastavením, mohou odpovědět opravdu jen vývojáři OS MikroTik RouterOS. Z pohledu vnější analýzy uvedeného protokolu lze vytvořit jednu parametrizovatelnou variantu protokolu (protokol MTP), která bude vyhovovat protokolům použitým v obou aplikacích. Proto již v dalším textu není pohlíženo na protokoly použité v obou aplikacích jako na dva různé protokoly, ale jako na jeden protokol s různým nastavením. V dalším textu je primárně popisován protokol použitý v aplikaci Terminal a v OS MikroTik RouterOS při komunikaci pomocí protokolu MT a na zjištěné odlišnosti chování protokolu použitého v aplikaci WinBox a v OS MikroTik RouterOS při komunikaci s aplikací WinBox protokolem MWB nebo MWBS je upozorněno.

Průběh komunikace je popisován na konfiguraci, kdy je zařízení typu server přímo propojeno s klientským zařízením pomocí standardní síťové technologie IEEE 802.3 Ethernet ve variantě 100Base-TX bez využití jakéhokoliv jejího rozšíření či nadstavby (např. VLAN tagging definované v IEEE 802.1q). Tato varianta síťové technologie byla zvolena na základě použité varianty síťové technologie v embedded systému, pro který je finální podoba komunikační knihovny určena. Pro popis komunikace je zvolen TCP/IP model komunikace (označovaný též jako Internet Model [4]), který je pro popis komunikace v TCP/IP sítích běžnější, než referenční ISO/OSI model komunikace. Komunikační modely popisující dále rozebírané typy komunikace jsou uvedeny na obrázku na následující stránce.



Obrázek 7.1: Komunikační modely popisovaných komunikací

Na následujícím obrázku je ukázáno složení ethernetových rámců s uvedením velikosti jednotlivých oblastí pro popisované typy komunikace:



Obrázek 7.2: Schéma složení ethernetových rámců

Z obrázku je patrné, že všechny tři případy komunikace využívají na vrstvě síťového rozhraní rámce typu Ethernet II. Rámec typu Ethernet II se skládá z hlavičky a z uživatelských dat, které následují ihned za jeho hlavičkou. Uživatelská data ethernetového rámce nesou data síťové vrstvy. Na síťové vrstvě je použit IP protokol verze 4 (IPv4). IP paket se skládá z hlavičky a z uživatelských dat. Uživatelská data IP paketu obsahují data transportní vrstvy. Na transportní vrstvě je použit protokol UDP. UDP datagram se dělí na hlavičku a uživatelská data. V uživatelských datech UDP datagramu jsou data aplikační vrstvy, která využívá protokol MTP. MTP segment je tvořen hlavičkou a uživatelskými daty. Uživatelská data MTP segmentu obsahují buď data protokolu MT nebo MWB nebo MWBS.

7.1 Vrstva síťového rozhraní

Na této vrstvě jsou použity rámce typu Ethernet II. Hlavička rámce tohoto typu má velikost 14 oktetů. Prvních 6 oktetů hlavičky (položka `dstethaddr`) je využito pro

MAC adresu síťového rozhraní příjemce rámce. Následujících 6 oktetů (položka `srcethaddr`) je využito pro MAC adresu síťového rozhraní odesílatele rámce. Oktety 13 a 14 (položka `type`) jsou využity pro kód protokolu neseného v uživatelských datech ethernetového rámce. Na 15. oktetu začínají uživatelská data rámce. Uživatelská data musí obsahovat min. 46 oktetů. Maximální počet oktetů tvořících uživatelská data je 1500. Struktura rámce typu Ethernet II je uvedena v následující tabulce:

bit	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	dstethaddr																															
32	dstethaddr																srcethaddr															
64	srcethaddr																															
96	type																data															

Tabulka 7.1: Struktura rámce typu Ethernet II

Názvy jednotlivých položek rámce typu Ethernet II jsou uvedeny v následující tabulce:

Označení položky	Název položky	Počet oktetů
`dstethaddr`	Destination Ethernet address	6
`srcethaddr`	Source Ethernet address	6
`type`	EtherType value	2
`data`	Data (payload)	46 – 1500

Tabulka 7.2: Názvy položek rámce typu Ethernet II

V případě komunikace z klientských aplikací Terminal a WinBox jsou rámce generované klientem vždy zasílány na ethernetovou broadcast adresu ff:ff:ff:ff:ff:ff, obsah položky `dstethaddr` je tedy vždy 0xFFFFFFFFFFFF. Uvedená ethernetová broadcast adresa je v souladu s IP broadcast adresou 255.255.255.255 použitou IP protokolem na síťové vrstvě. Zařízení s OS MikroTik RouterOS zasílá rámce vždy přímo na MAC adresu cílového síťového rozhraní. Takto tomu je v situacích, kdy zařízení s OS MikroTik RouterOS pracuje jako server při komunikaci s aplikací WinBox, jako server při komunikaci s aplikací Terminal, jako server při komunikaci s jiným zařízením s OS MikroTik RouterOS pomocí protokolu MT, ale i v situacích, kdy komunikuje jako klient vůči jinému zařízení s OS MikroTik RouterOS pomocí protokolu MT. Zjednodušeně lze tedy říci, že pokud je rámec odeslán z PC, tak je odeslán na ethernetovou broadcast adresu ff:ff:ff:ff:ff:ff. V případě, že je rámec odeslán ze zařízení s OS MikroTik RouterOS, tak je zaslán přímo na MAC adresu cílového síťového rozhraní. Rámce komunikace mezi dvěma zařízeními s OS MikroTik

RouterOS jsou tedy vždy adresovány přímo na příslušnou MAC adresu cílového síťového rozhraní. U rámců odesílaných zařízením s OS MikroTik RouterOS je potom ovšem v nesouladu použitá IP broadcast adresa 255.255.255.255 IP protokolem na síťové vrstvě a použitá cílová ethernetová adresa. V tomto případě správně na síťové vrstvě IP broadcast adrese 255.255.255.255 odpovídá na vrstvě síťového rozhraní ethernetová broadcast adresa ff:ff:ff:ff:ff:ff. Uvedený nesoulad je nejspíše důvodem, proč aplikace Terminal a WinBox tento způsob naplnění položky `dstethaddr` nevyužívají a odchozí ethernetové rámce mají v této položce hodnotu 0xFFFFFFFFFFFF. V tomto případě je možné bez problému využít standardního datagramového soketu pro protokol UDP. V opačném případě by bylo nutné použít soket typu raw a sestavení ethernetového rámce a protokoly IPv4 a UDP implementovat až na úrovni kódu samotné aplikace. Tomu se pravděpodobně vývojáři aplikací Terminal a WinBox chtěli vyhnout. Toto řešení ale zbytečně způsobuje větší zatížení sítě, protože ethernetové rámce komunikace jsou zasílány na všechna zařízení v daném segmentu sítě. Dalším aspektem je nižší bezpečnost tohoto řešení.

V následující tabulce je uveden přehled možných variant komunikací proti zařízení s OS MikroTik RouterOS a příslušný způsob naplnění položky `dstethaddr`:

		Cílová ethernetová adresa	
Klient	**Protokol**	**Klient**	**Server**
Terminal	MT	ff:ff:ff:ff:ff:ff	MAC
RouterOS	MT	MAC	MAC
WinBox	MWB/MWBS	ff:ff:ff:ff:ff:ff	MAC
Terminal *	MT	ff:ff:ff:ff:ff:ff	00:00:00:00:00:00
WinBox*	MWB/MWBS	ff:ff:ff:ff:ff:ff	ff:ff:ff:ff:ff:ff

Tabulka 7.3: Cílové ethernetové adresy vyskytující se v rámcích

Klienti, kteří jsou označeni hvězdičkou, jsou provozováni v operačním systému Microsoft Windows bez nainstalovaného síťového klienta nazvaného „Klient sítě Microsoft“. Změna na straně chování serveru je způsobena tím, že daný klient nedokáže bez nainstalované komponenty detekovat MAC adresu síťového rozhraní PC, na kterém je spuštěn. Klient Terminal zašle protokolem MTP jako MAC adresu síťového rozhraní, na kterém je spuštěn ethernetovou adresu 00:00:00:00:00:00. Server tuto adresu použije v ethernetovém rámci jako MAC adresu cílového síťového rozhraní. Síťové rozhraní, které by mělo být správně adresováno, pochopitelně tento rámec zahodí, protože ethernetová adresa v ethernetovém rámci se neshoduje s MAC adresou daného síťového rozhraní PC, na kterém běží aplikace Terminal a tato aplikace nemá přepnutou síťovou kartu do promiskuitního režimu, aby byla schopna v této situaci takovýto rámec

přijmout. Klient WinBox se s tímto nestandardním stavem vyrovná lépe. Pokud se mu nepodaří úspěšně detekovat MAC adresu síťového rozhraní, přes které má komunikovat, použije ethernetovou broadcast adresu ff:ff:ff:ff:ff:ff, kterou vloží do MTP protokolu, jako MAC adresu síťového rozhraní pomocí kterého komunikuje. Tuto ethernetovou adresu potom server zpětně použije jako ethernetovou adresu příjemce rámce. Z uvedené situace je patrné, že server pro naplnění položky cílové ethernetové adresy v ethernetovém rámci používá adresu získanou z MTP protokolu a nikoliv adresu z rámce, ze které rámec přišel.

V položce `srcethaddr` je vždy MAC adresa síťového rozhraní, ze které byl ethernetový rámec odeslán. Tuto položku vyplní odesílající strana na základě znalosti MAC adresy síťového rozhraní, ze kterého bude ethernetový rámec odeslán.

V položce `type` je vždy jak u rámce odeslaného klientem, tak i u rámce odeslaného serverem, použita EtherType hodnota 0x0800, která určuje, že v uživatelských datech ethernetového rámce je obsažen IP protokol verze 4.

7.2 Síťová vrstva

Na této vrstvě je pravděpodobně použit standardizovaný IP protokol verze 4 (IPv4), definovaný v RFC 791 [5]. Hlavička IP paketu tohoto protokolu je v popisované komunikaci tvořena 20 oktety. Vyšší čtyři bity prvního oktetu (položka `ver`) obsahují verzi IP protokolu. Nižší čtyři bity prvního oktetu (položka `hlen`) obsahují délku hlavičky IP paketu v počtu násobků čtyřoktetových slov (násobků 32 bitů). Hodnota ve vyšších šesti bitech druhého oktetu (položka `dscp`) definuje tzv. bod DSCP, umožňující přiřazení různých úrovní služeb síťovému provozu. Podrobnosti jsou uvedeny v RFC 2474 [6]. Dva nejnižší bity (položka `ecn`) druhého oktetu slouží pro uložení příznaku o schopnosti vyhodnocovat přetížení sítě a příznaku o aktuálním stavu přetížení sítě. Podrobnosti jsou uvedeny v RFC 3168 [7]. Oktety 3 a 4 (položka `tlen`) obsahují počet oktetů, které obsahuje celý IP datagram. Oktety 5 a 6 (položka `id`) obsahují jednoznačný identifikátor IP datagramu, který je využíván zejména při defragmentaci určitého IP datagramu. Nejvyšší tři bity sedmého oktetu (položka `flags`) tvoří příznakové pole pro řízení fragmentace IP datagramu. Nižších pět bitů sedmého oktetu a osmý oktet (položka `foffset`) obsahuje offset daného IP fragmentu v původním IP datagramu v jednotkách počtu bloků tvořených osmi oktety. Devátý oktet (položka `ttl`) obsahuje hodnotu udávající zbývající dobu života IP paketu. Desátý oktet (položka `proto`) obsahuje kód protokolu vyšší vrstvy, kterému mají být předána uživatelská data obsažená v IP datagramu. Oktety 11 a 12 (položka `hchksum`) obsahují kontrolní součet hlavičky IP paketu. Algoritmus výpočtu kontrolního součtu je definován v RFC 1071 [8]. Oktety 13 až 16 (položka `srcaddr`) jsou využity pro uložení IP adresy odesílatele IP paketu. Oktety 17 až 20 (položka `dstaddr`) jsou využity pro uložení IP

adresy příjemce IP paketu. Na 21. oktetu začínají uživatelské oktety. Struktura popisovaného IP paketu je uvedena v následující tabulce:

bit	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	ver				hlen				dscp						ecn		tlen															
32	id																flags				foffset											
64	ttl								proto								hchksum															
96	srcaddr																															
128	dstaddr																															
160	data																															

Tabulka 7.4: Struktura IPv4 paketu

Názvy jednotlivých položek IPv4 paketu jsou uvedeny v následující tabulce:

Označení položky	Název položky	Počet oktetů
ver	Version	$^1/_2$
hlen	Header length	$^1/_2$
dscp	Differentiated services code point	$^3/_4$
ecn	Explicit Congestion Notification	$^1/_4$
tlen	Total length	2
id	Identification	2
flags	Flags	$^3/_8$
foffset	Fragment offset	$^{13}/_8$
ttl	Time to live	1
proto	Protocol	1
hchksum	Header checksum	2
srcaddr	Source address	4
dstaddr	Destination address	4
data	Data (payload)	-

Tabulka 7.5: Názvy položek IPv4 paketu

Položka `ver` je pro všechny popisované typy komunikací a pro oba směry komunikace naplněna vždy hodnotou 4, protože je použit IP protokol verze 4.

Položka `hlen` je vždy naplněna hodnotou 5, protože se využívá standardní délka hlavičky IP protokolu (20 oktetů) a rozšiřující pole hlavičky nejsou použity.

Položka `dscp` je vždy naplněna hodnotou 0, což označuje běžný síťový provoz bez jakéhokoliv zvýhodnění oproti ostatnímu síťovému provozu.

Položka `ecn` obsahuje vždy hodnotu 0, což označuje, že informování koncových účastníků přenosu o přetížení některé sítě na přenosové cestě není podporováno.

Položka `tlen` obsahuje celkovou délku IP datagramu. Vzhledem k tomu, že zařízení s OS MikroTik RouterOS ani aplikace Terminal a WinBox samy negenerují IP datagramy větší než je hodnota MTU daného síťového rozhraní, je tato hodnota zároveň délkou IP paketu. Vzhledem k tomu, že popisovaná komunikace je realizovatelná pouze v rámci daného segmentu sítě, je fragmentace na přenosové trase prakticky vyloučena.

Položka `id` obsahuje jednoznačný identifikátor IP datagramu, který se využívá zejména k jednotnému označení fragmentů jednoho IP datagramu. Hodnoty jsou generovány obvyklým způsobem – postupným inkrementováním hodnoty získané z příchozího IP paketu.

Příznakové pole `flags` je ve všech analyzovaných IP paketech nulové, z čehož plyne, že nedošlo na přenosové cestě k fragmentaci. Klienti ani server tedy sami fragmentované IP datagramy neprodukují, ale fragmentace IP datagramů není klientem ani serverem explicitně zakázána.

Položka `foffset` má ve všech analyzovaných IP paketech nulovou hodnotu, protože u žádného z IP datagramů nedošlo k jeho fragmentaci. Jak již bylo zmíněno, vzhledem k tomu, že komunikace je využívána pouze v rámci jednoho segmentu sítě a účastníci komunikace sami fragmentované IP datagramy negenerují, lze očekávat v této položce prakticky vždy nulovou hodnotu. V opačném případě by položka obsahovala offset daného fragmentu IP datagramu v původním IP datagramu. Hodnota je udávána jako násobek bloku osmi oktetů.

Položka `ttl` udává zbývající životnost daného IP paketu. Zařízení s OS MikroTik RouterOS nastavují hodnotu této položky na 64. Aplikace Terminal a Winbox nastavují hodnotu této položky na 128, což je výchozí hodnota operačního systému Microsoft Windows.

Položka `proto` vždy obsahuje hodnotu 17, protože IP datagram nese vždy ve svých uživatelských datech data protokolu UDP.

Položka `hchksum` obsahuje hodnotu kontrolního součtu hlavičky IP paketu. Stejné hodnoty jako se vyskytují v této položce lze získat aplikací algoritmu uvedeného v RFC 1071 [8].

Položka `srcaddr` je vyplněna v případě klientů Terminal a WinBox hodnotou odpovídající IP adrese, která je danému rozhraní PC, ze kterého se komunikuje, přidělena. IP pakety odeslané zařízením s OS MikroTik RouterOS mají tuto položku nulovou (0x00000000), což odpovídá IP adrese 0.0.0.0.

Položku `dstaddr` vyplňují jak všichni dostupní klienti, tak i servery na hodnotu 0xFFFFFFFF, čemuž odpovídá IP adresa plného broadcastu 255.255.255.255. Zde je třeba upozornit na to, že server tuto hodnotu položky vyžaduje. V opačném případě, kdy je v této položce např. hodnota odpovídající IP adrese konfigurovaného zařízení nebo je použita IP broadcast adresa dané podsítě (např. 192.168.88.255/24), server tyto IP pakety ignoruje.

7.3 Transportní vrstva

Na této vrstvě je použit protokol, který se jeví, že vyhovuje standardizovanému protokolu UDP, definovaného v RFC 768 [9]. Hlavičku takovéhoto UDP datagramu tvoří 8 oktetů. První dva oktety (položka `srcprt`) označují UDP port na straně odesílatele UDP datagramu, ze kterého byl UDP datagram odeslán. Následující dva oktety (položka `dstprt`) označují cílový port na straně příjemce UDP datagramu, pro který je UDP datagram určen. Oktety 5 a 6 (položka `length`) určují délku celého UDP datagramu včetně jeho hlavičky. Oktety 7 a 8 (položka `chksum`) obsahují kontrolní součet vypočítaný z celého UDP datagramu. Na 9. oktetu UDP datagramu začínají uživatelská data UDP datagramu. Struktura UDP datagramu je uvedena v následující tabulce:

bit	00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0	srcprt	dstprt
32	length	chksum
64	data	

Tabulka 7.6: Struktura UDP datagramu

Názvy položek UDP datagramu s uvedením jejich velikostí jsou uvedeny v následující tabulce:

Označení položky	Název položky	Počet oktetů
`srcprt`	Source port	2
`dstprt`	Destination port	2
`length`	Length	2
`chksum`	Checksum	2
`data`	Data (payload)	-

Tabulka 7.7: Názvy položek UDP datagramu

MTP server v zařízení s OS MikroTik RouterOS naslouchá na UDP portu 20561. UDP datagramy mohou být serveru klientem zasílány z libovolného zdrojového UDP portu. Aplikace Terminal i klient v OS MikroTik RouterOS při komunikaci pomocí MT protokolu ale odesílají komunikaci vždy z portu 20561. Aplikace WinBox zasílá komunikaci z UDP portu operačním systémem jí přiděleného. Při každém spuštění aplikace WinBox je tedy UDP port, ze kterého tato aplikace odesílá UDP datagramy různý.

Položka `srcprt` obsahuje hodnotu 20561 v případě odeslání UDP datagramu z aplikace Terminal nebo při odeslání UDP datagramu ze zařízení s OS MikroTik RouterOS, které může být jak ve funkci klienta, tak i ve funkci serveru. Pokud se jedná o UDP datagram odeslaný z aplikace WinBox, odpovídá tato hodnota přidělenému portu aplikaci operačním systémem (> 1024).

Položka `dstprt` obsahuje hodnotu 20561 v případě odeslání UDP datagramu aplikaci Terminal nebo zařízení s OS MikroTik RouterOS, které může být jak ve funkci klienta, tak i ve funkci serveru. Pokud se jedná od UDP datagram odeslaný zařízením s OS MikroTik RouterOS aplikaci WinBox, je tato hodnota totožná s hodnotou portu, ze kterého zahájila aplikace WinBox komunikaci.

Položka `length` je naplněna počtem oktetů, které tvoří celý UDP datagram. Minimální hodnota této položky je tedy 8 (délka hlavičky UDP datagramu). Vezmeme-li v úvahu, že na aplikační vrstvě je použit MTP protokol, jehož délka hlavičky je 22 oktetů, lze říci, že hodnota položky `length` musí být minimálně 30.

Položka `chksum` je naplněna hodnotou kontrolního součtu, který jak bylo ověřeno, lze získat aplikací algoritmu popsaného v RFC 768 [9].

7.4 Aplikační vrstva

Na této vrstvě je použit nedokumentovaný proprietární protokol, který byl v průběhu analýzy označen jako MikroTik Transmission Protocol (MTP). Tento protokol nese buď data protokolu MikroTik Terminal (MT) nebo data protokolu MikroTik WinBox (MWB) nebo data protokolu MikroTik WinBox Secure (MWBS). Hlavička MTP segmentu má pevně danou velikost (22 oktetů). První oktet (položka `xconst`) představuje neznámou konstantu. Druhý oktet (položka `sgtype`) definuje typ MTP segmentu. Následujících 6 oktetů (položka `srcmac`) je použito pro MAC adresu síťového rozhraní, ze kterého byl MTP segment odeslán. Dalších 6 oktetů (položka `dstmac`) je použito pro MAC adresu síťového rozhraní, pro které je MTP segment určen. Oktety 15 až 18 (položka `sessid`) tvoří identifikační číslo jednoznačně označující dané spojení (session) mezi dvěma účastníky. Oktety 19 až 22 (položka `dcount`) obsahují počet doposud odeslaných/přijatých uživatelských oktetů, případně jsou pro některé typy MTP segmentů tyto oktety nulové. Na 23. oktetu začínají

uživatelská data (pokud jsou obsažena). Struktura MTP segmentu je uvedena v následující tabulce:

bit	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	xconst								sgtype								srcmac															
32	srcmac																															
64	dstmac																															
96	dstmac																sessid															
128	sessid																dcount															
160	dcount																data															

Tabulka 7.8: Struktura MTP segmentu

Názvy položek MTP segmentu s uvedením jejich velikostí jsou uvedeny v následující tabulce:

Označení položky	Název položky	Počet oktetů
xconst	Unknown constant	1
sgtype	Segment type	1
srcmac	Source MAC address	6
dstmac	Destination MAC address	6
sessid	Session ID number	4
dcount	Data count	4
data	Data (payload)	-

Tabulka 7.9: Názvy položek MTP segmentu

Položka `xconst` obsahuje vždy identickou hodnotu 0x01 ve všech odchycených MTP segmentech při využití všech dostupných dvojic klient-server aplikací.

Hodnota v položce `sgtype` definuje typ segmentu. Pro otevření spojení (session) se používá segment typu OPEN, jenž má v položce `sgtype` hodnotu 0x00. Pro přenos dat protokolu použitého ve vyšší vrstvě komunikačního modelu se používá segment typu DATA, jenž má v položce `sgtype` hodnotu 0x01. Segment typu ACK, který potvrzuje přijatý segment typu OPEN nebo DATA má v položce `sgtype` hodnotu 0x02. Pro ukončení spojení (session) se používá segment typu CLOSE, jenž má v položce `sgtype` hodnotu 0xFF. Na segmenty s jinou hodnotou v položce `sgtype` než 0x00, 0x01, 0x02 nebo 0xFF MTP server nijak nereaguje. Seznam používaných typů segmentů je uveden v tabulce na následující straně.

Hodnota	Označení	Název segmentu
0x00	OPEN	Open new session
0x01	DATA	Data
0x02	ACK	Acknowledgement
0xFF	CLOSE	Close session

Tabulka 7.10: Typy MTP segmentů (hodnoty položky `sgtype`)

Položka `srcmac` obsahuje MAC adresu síťového rozhraní, ze kterého odesílatel MTP segment odeslal. Každý oktet reprezentuje příslušný oktet MAC adresy síťového rozhraní odesílatele MTP segmentu. První oktet položky `srcmac` tedy odpovídá prvnímu oktetu MAC adresy síťového rozhraní. Obsah položky `srcmac` se používá pro identifikaci odesílatele MTP segmentu na úrovni MTP protokolu.

Položka `dstmac` obsahuje MAC adresu síťového rozhraní příjemce, pro nějž je MTP segment určen. Souvislost oktetů položky `dstmac` s oktety MAC adresy je obdobná jako u položky `srcmac`. Obsah položky `dstmac` se používá pro identifikaci příjemce MTP segmentu na úrovni MTP protokolu.

Význam jednotlivých oktetů v položce `sessid` se liší podle toho, kdo je odesílatelem MTP segmentu. V případě, že je odesílatelem MTP segmentu klient, je v prvních dvou oktetech (dílčí položka `sessnm`) obsaženo identifikační číslo spojení (session) a ve zbylých dvou oktetech (dílčí položka `sesspr`) je kód komunikačního protokolu, který je obsažen v uživatelských datech MTP segmentu. V situaci, kdy je odesílatelem server, je význam těchto dvou dvojic oktetů prohozen. Vše je ukázáno v následujících dvou tabulkách:

bit	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	sessnm																sesspr															

Tabulka 7.11: Struktura položky `sessid` v MTP segmentu odeslaného klientem

bit	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	sesspr																sessnm															

Tabulka 7.12: Struktura položky `sessid` v MTP segmentu odeslaného serverem

Dílčí položka `sessnm` může nabývat libovolné hodnoty (0x0000 až 0xFFFF). Hodnota pouze nesmí kolidovat s případně již otevřeným spojením (session) mezi uvažovanými dvěmi zařízeními. Hodnotu této položky vygeneruje klient při odesílání MTP segmentu

typu OPEN. V průběhu komunikace pomocí daného spojení (session) zůstává hodnota položky `sessnm` konstantní. Klient Terminal hodnotu této položky generuje náhodně. Nejvyšší čtyři bity dílčí položky `sessnm` jsou ale vždy nulové. Klient WinBox generuje hodnotu této položky na základě časového údaje v sekundách. Při generování požadavků k otevření spojení (session) každou 1 sekundu, se hodnota této položky periodicky zvětšuje o jedničku. Klient pro protokol MT integrovaný v OS MikroTik RouterOS zvyšuje hodnotu této položky o jedničku u každého nového požadavku k otevření nového spojení (session) nezávisle na čase.

V dílčí položce `sesspr` se vyskytují pouze dva kódy protokolů obsažených v uživatelských oktetech MTP segmentu. V případě, že je v uživatelských datech MTP segmentu obsažen MT protokol, je v dílčí položce `sesspr` hodnota 0x0015. Protokoly MWB a MWBS označuje kód 0x0F90.

Hodnota	Protokol
0x0015	MikroTik Terminal
0x0F90	MikroTik WinBox, MikroTik WinBox Secure

Tabulka 7.13: Kódy protokolů vyšší vrstvy (hodnoty položky `sesspr`)

Přesný význam položky `dcount` závisí na typu MTP segmentu. U segmentu typu OPEN je hodnota v této položce vždy nulová (0x00000000). U segmentu typu DATA je v této položce uložen součet doposud odeslaných uživatelských oktetů od otevření spojení (session). Uživatelské oktety obsažené v daném MTP segmentu typu DATA započteny nejsou. V případě potvrzovacího segmentu typu ACK obsahuje položka `dcount` součet všech přijatých uživatelských oktetů, které příjemce potvrzuje, že je přijal. V případě segmentu typu CLOSE pro ukončení spojení (session), je obsah položky opět nulový (0x00000000).

Uživatelské oktety (pole data) obsahuje pouze MTP segment typu DATA. MTP segmenty typu OPEN, ACK a CLOSE uživatelské oktety neobsahují.

Princip funkce protokolu MikroTik Transmission Protocol

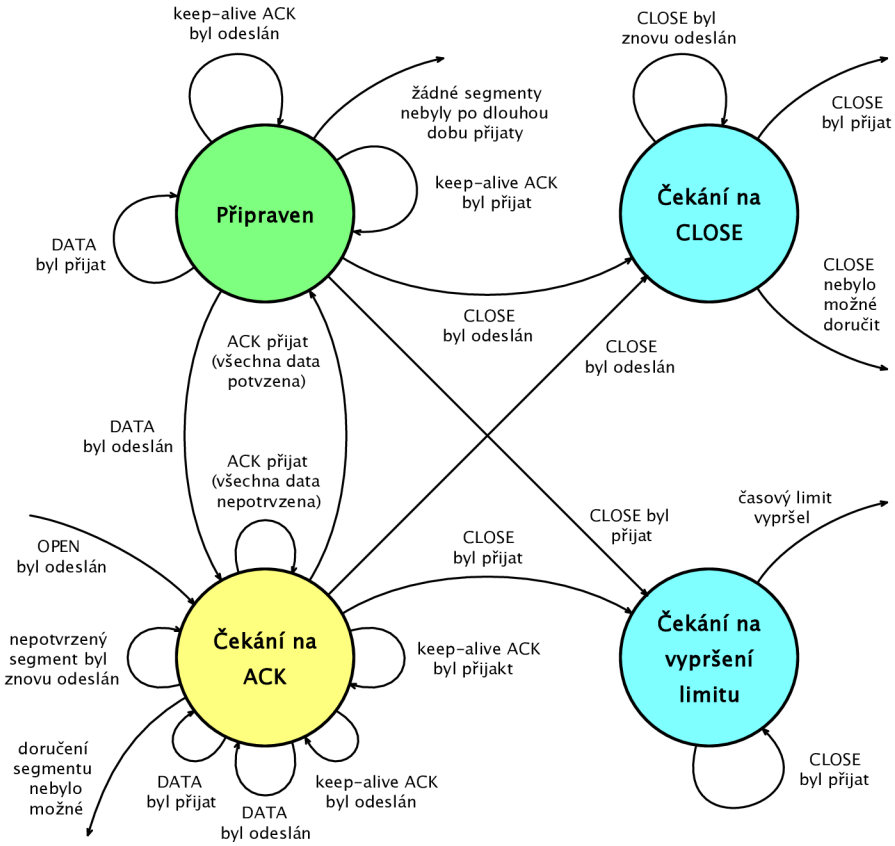
Nové spojení se otevírá MTP segmentem typu OPEN, který nenese žádné uživatelské oktety. MTP segment tohoto typu zasílá klient. Pokud server akceptuje požadavek na otevření nového spojení s danými parametry, potvrdí příjem tohoto MTP segmentu zasláním MTP segmentu typu ACK. Tímto je spojení otevřeno. Klient může nyní zasílat data vyššího protokolu pomocí MTP segmentů typu DATA. Pokud klient neodešle první MTP segment do několika jednotek sekund, spojení je na straně serveru ukončeno a server již neakceptuje žádné MTP segmenty vztahující se k danému spojení. Při takovémto ukončení spojení není serverem odeslán žádný MTP segment.

Každý MTP segment typu DATA přijatý příjemcem musí být odesílateli potvrzen zasláním správného MTP segmentu typu ACK. Pokud není do určité doby MTP segment typu DATA odesílateli tohoto segmentu tento segment potvrzen správným MTP segmentem typu ACK, odesílatel opakuje jeho odeslání. Odesílatel MTP segmentu může toto opětovné odeslání po vypršení časového limitu znovu zopakovat. Počet přeposlání daného typu MTP segmentu je ale omezen. V případě, že vyprší stanovený limit pro příchod MTP segmentu typu ACK a počet dovolených přeposlání MTP segmentu je vyčerpán, je spojení na straně, u které došlo k popisované události, ukončeno. Žádný MTP segment již není dále touto stranou odeslán (např. o ukončení spojení) ani akceptován. Počty opakování přeposlání nepotvrzeného MTP segmentu typu DATA se liší podle použitého serveru/klienta. Délka intervalu, který je vyhrazen na potrvzení odeslaného segmentu typu DATA je závislá na počtu již přeposlaných MTP segmentů a konkrétní hodnoty se opět liší podle použitého serveru/klienta.

V případě použití MTP protokolu pro nesení dat protokolu MT, server i klient od okamžiku úspěšného otevření spojení odesílají opakovaně s periodou 10 sekund MTP segmenty typu ACK, které obsahují v položce `dcount` počet přijatých uživatelských oktetů od protistrany v segmentech typu DATA. Tyto MTP segmenty typu ACK jsou nazvány keep-alive, protože se nevztahují k žádnému segmentu typu DATA. Keep-alive MTP segment typu ACK je tedy totožný s posledním odeslaným MTP segmentem typu ACK. Perioda 10 sekund je dodržena vždy, nezávisle na tom, zda-li byl nebo nebyl od posledního keep-alive segmentu typu ACK odeslán MTP segment typu DATA nebo ACK. Keep-alive MTP segmenty typu ACK slouží pro detekci výpadku spojení v případě, kdy nejsou zasílány žádné MTP segmenty typu DATA. Pokud by nebyly použity keep-alive MTP segmenty typu ACK, nebylo by možné zjistit výpadek spojení v době, kdy se nepřenášejí žádné MTP segmenty. Spojení by tak bylo ukončeno až při pokusu odeslat MTP segment typu DATA. V situaci s použitými keep-alive MTP segmenty typu ACK dojde k ukončení spojení, pokud není přijat žádný MTP segment po určitý stanovený limit (20 nebo 30 sekund). Délka tohoto časového limitu se liší podle použitého klienta/serveru. Při použití MTP protokolu pro nesení dat protokolu MWB nebo MWBS keep-alive MTP segmenty typu ACK na straně klienta generovány nejsou, ale na straně serveru ano.

Spojení se regulérně ukončuje v případě použitého protokolu MT zasláním MTP segmentu typu CLOSE serverem nebo klientem. Protistrana obdržený MTP segment typu CLOSE potvrdí MTP segmentem typu CLOSE. MTP segment typu CLOSE neobsahuje žádné uživatelské oktety. V případě použití protokolu MWB nebo MWBS se ukončení pomocí segmentu typu CLOSE nevyužívá.

Celé stavové chování MTP protokolu na straně klienta je detailně uvedeno ve stavovém diagramu na následující straně.



Obrázek 7.3: Stavový diagram MTP protokolu na straně klienta

Do stavového diagramu jsou zahrnuty jen události, které vyvolají nějakou skrytou akci (např. úpravu stavových proměnných protokolu). Události, které nemají na celý stav protokolu naprosto žádný vliv (např. příchod neplatného MTP segmentu), zahrnuty nejsou. Z předchozího popisu chování MTP protokolu je zřejmé, že všechny uvedené události nepředstavují ve skutečnosti pouze samotnou událost, ale i průchod stavem, ve kterém se realizují potřebné akce odpovídající vzniknuté události. Např. událost označující příjem platného MTP segmentu typu DATA způsobí přechod do stavu, ve kterém je vygenerován a odeslán potvrzující MTP segment typu ACK a je provedeno předání uživatelských oktetů ke zpracování vyššímu protokolu a je provedena úprava vnitřních stavových proměnných MTP protokolu. Tyto stavy by však tento přehledový stavový diagram velmi znepřehlednily.

Dále je třeba upozornit, že uvedený stavový diagram MTP protokolu je souhrnný pro všechny analyzované komunikace. Z toho plyne, že ne všechny jeho části musí být použity současně. Stavový diagram v kompletní podobě, tak jak je uveden, je využíván pouze u klientů komunikujících pomocí protokolu MT. Toto ale neznamená újmu na obecnosti stavového diagramu MTP protokolu, protože na tyto nevyužité části lze pohlížet tak, že jsou s aktuálně použitým vyšším přenosovým protokolem vypnuté. Např. zasílání keep-alive MTP segmentů typu ACK je u klientů komunikujících pomocí protokolů MWB nebo MWBS vypnuto, ale příjem těchto segmentů je aktivní.

V několika následujících tabulkách jsou uvedeny časové limity, během kterých odesílatel čeká na potvrzení odeslaného MTP segmentu vybraného typu. Poslední interval je vždy neznámý, protože jej nelze přímo změřit, protože při jeho vypršení již nedojde k opětovnému odeslání MTP segmentu daného typu, ale spojení je ukončeno.

Časové intervaly, během nichž musí server potvrdit klientem odeslaný MTP segment typu OPEN, aby klient tento MTP segment typu OPEN znovu neodeslal nebo neukončil pokus o otevření spojení, jsou uvedeny v následující tabulce:

Klient	Protokol	Časové intervaly [ms]						
Terminal	MT	100	200	400	800	?		
RouterOS	MT	20	30	50	90	170	330	?
WinBox	MWB	100	100	?				

Tabulka 7.14: Časové intervaly vyhrazené pro potvrzení MTP segmentu typu OPEN serverem

Časové intervaly, během nichž musí server potvrdit klientem odeslaný MTP segment typu DATA, aby klient tento MTP segment typu DATA znovu neodeslal nebo neukončil otevřené spojení, jsou uvedeny v následující tabulce:

Klient	Protokol	Časové intervaly [ms]							
Terminal	MT	100	100	200	400	800	?		
RouterOS	MT	20	20	30	50	90	170	330	?
WinBox	MWB	31x 100 ms							

Tabulka 7.15: Časové intervaly vyhrazené pro potvrzení MTP segmentu typu DATA serverem

Časové intervaly, během nichž musí klient potvrdit serverem odeslaný MTP segment typu DATA, aby server tento MTP segment typu DATA znovu neodeslal nebo neukončil otevřené spojení, jsou uvedeny v následující tabulce:

Server	Protokol	Časové intervaly [ms]							
RouterOS	MT	20	20	30	50	90	170	330	?
RouterOS	MWB	20	20	30	50	90	170	330	?

Tabulka 7.16: Časové intervaly vyhrazené pro potvrzení MTP segmentu typu DATA klientem

Časové intervaly, během nichž musí server potvrdit klientem odeslaný MTP segment typu CLOSE, aby klient tento MTP segment typu CLOSE znovu neodeslal nebo neukončil otevřené spojení, jsou uvedeny v následující tabulce:

Klient	Protokol	Časové intervaly [ms]						
Terminal	MT	100	200	?				

Tabulka 7.17: Časové intervaly vyhrazené pro potvrzení MTP segmentu typu CLOSE serverem

Časové intervaly, během nichž musí klient potvrdit serverem odeslaný MTP segment typu CLOSE, aby server tento MTP segment typu CLOSE znovu neodeslal nebo neukončil otevřené spojení, jsou uvedeny v následující tabulce:

Server	Protokol	Časové intervaly [ms]						
RouterOS	MT	20	30	50	90	170	?	

Tabulka 7.18: Časové intervaly vyhrazené pro potvrzení MTP segmentu typu CLOSE klientem

7.4.1 MikroTik Terminal

Jedná se o nedokumentovaný proprietární protokol, který byl v průběhu analýzy pojmenován jako MikroTik Terminal (MT). Tento protokol nemá žádnou hlavičku, jedná se o tok dat ze/do vzdáleného systému, ve kterém se navíc vyskytují jen speciální řídicí sekvence (`ctrlsq`).

Hlavička řídicí sekvence je složena z 9 oktetů. V prvních pěti oktetech (položka `sqtype`) je uložen identifikátor dané řídicí sekvence, který určuje typ řídicí sekvence. Následující 4 oktety (položka `length`) jsou použity pro vyjádření počtu uživatelských oktetů v řídicí sekvenci. Na 10. oktetu začínají uživatelské oktety řídicí sekvence (pokud jsou použity). Ne všechny řídicí sekvence oblast uživatelských oktetů využívají. Struktura řídicí sekvence je uvedena v následující tabulce:

bit	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	sqtype																															
32	sqtype																length															
64	length																data															

Tabulka 7.19: Struktura řídicí sekvence (`ctrlsq`)

Názvy položek řídicí sekvence s uvedením jejich velikostí jsou uvedeny v následující tabulce:

Označení položky	Název položky	Počet oktetů
`sqtype`	Sequence type	5
`length`	Data length	4
`data`	Data (payload)	`length`

Tabulka 7.20: Názvy položek řídicí sekvence

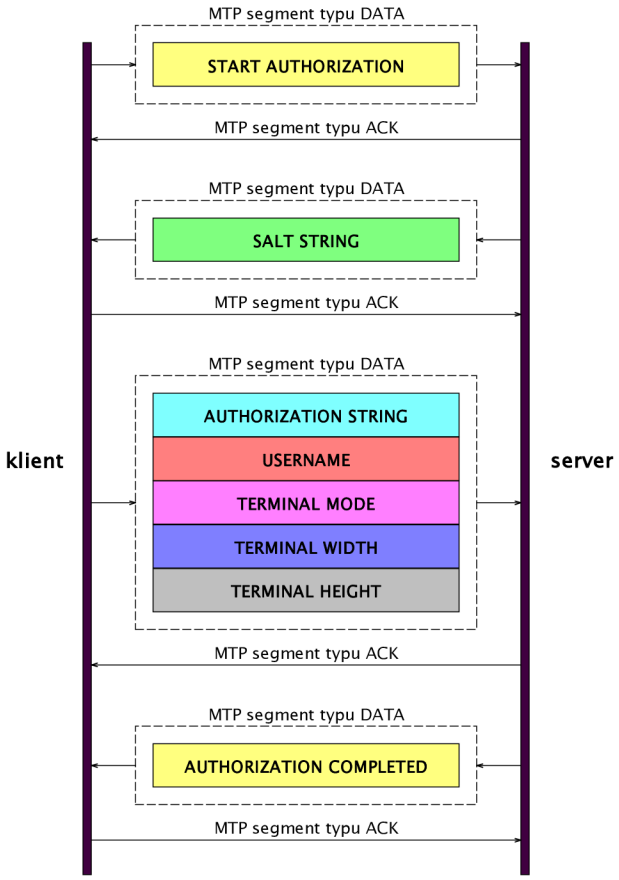
Obsah položky `sqtype` klient nebo server vyplní na základě akce, kterou chce provést. Obsah položky `length` vyplní na základě znalosti počtu uživatelských oktetů vztahujících se k dané řídicí sekvenci. Oblast `data` vyplní klient nebo server na základě pevně daného algoritmu vztahujícího se k danému typu řídicí sekvence. Zjištěné typy řídicích sekvencí, jejich identifikátory, zavedené značení a typický počet uživatelských oktetů, které řídicí sekvence daného typu obsahuje, jsou uvedeny v následující tabulce:

Hodnota	Název řídicí sekvence	Typ. poč. uživ. oktetů
0x563412FF00	START AUTHORIZATION	0
0x563412FF01	SALT STRING	16
0x563412FF02	AUTHORIZATION STRING	17
0x563412FF03	USERNAME	-
0x563412FF04	TERMINAL MODE	5
0x563412FF05	TERMINAL WIDTH	2
0x563412FF06	TERMINAL HEIHGT	2
0x563412FF07	AUTHORIZATION ERROR	0
0x563412FF09	AUTHORIZATION COMPLETED	0

Tabulka 7.21: Kódy řídicích sekvencí (položka `sqtype`)

Řídicí sekvence nazvaná START AUTHORIZATION je klientem použita pro informování serveru o započetí procesu autorizace ke vzdálenému terminálu. Obsah položky `length` je u tohoto typu řídicí sekvence nulový (0x00000000), žádné uživatelské oktety tedy nejsou v řídicí sekvenci použity. Server na základě přijetí této řídicí sekvence odešle zpět klientovi řídicí sekvenci nazvanou SALT STRING. Sekvence SALT STRING obsahuje náhodný řetězec, který je potřebný pro výpočet autorizačního řetězce. Náhodný řetězec je typicky tvořen 16ti oktety. Klient na základě obdrženého náhodného řetězce vypočítá autorizační řetězec, který je součástí řídicí

sekvence nazvané AUTHORIZATION STRING. Tento řetězec je složen ze znaku s ASCII hodnotou 0x00 a MD5 haše řetězce, který je složen ze znaku s ASCII hodnotou 0x00, řetězce obsahujícího přihlašovací heslo, náležející k příslušnému uživatelskému jménu a přijatého náhodného řetězce. Řídicí sekvence označená jako USERNAME obsahuje v uživatelských oktetech řetězec, který je tvořen znaky uživatelského jména, pomocí kterého je požadováno se k systému autorizovat. Řídicí sekvence TERMINAL MODE nese ve všech analyzovaných komunikacích pět uživatelských oktetů, které obsahují znaky z řetězece „linux“. Řídicí sekvence TERMINAL WIDTH obsahuje dva uživatelské oktety, jejichž hodnota tvoří počet znaků na řádek obrazovky terminálu. Řídicí sekvence TERMINAL HEIGHT obsahuje také dva oktety, jenž tvoří počet řádků obrazovky terminálu. U řídicích sekvencí TERMINAL WIDTH a TERMINAL HEIGHT je třeba upozornit na obrácené pořadí oktetů, kdy nejméně významný oktet je uveden v pořadí jako první, oproti všem ostatním položkám, kdy je uveden jako poslední. Řídicí sekvence AUTHORIZATION STRING, USERNAME, TERMINAL MODE, TERMINAL WIDHT a TERMINAL HEIGHT odešle klient v uvedeném pořadí společně v jednom MTP segmentu typu DATA. Řídicí sekvenci AUTHORIZATION ERROR odešle server v případě přijetí nekonzistentních dat a současně s tím ukončí spojení odesláním MTP segmentu typu CLOSE. V případě úspěšného dokončení autorizačního procesu, odešle server řídicí sekvenci AUTHORIZATION COMPLETED. Tato řídicí sekvence neobsahuje žádné uživatelské oktety. Úspěšné ukončení autorizačního procesu ale nijak nesouvisí s jeho výsledkem (přístup umožněn nebo zamítnut). V případě umožnění přístupu ke vzdálenému terminálu je serverem v následujících datech odeslán výpis obrazovky vzdáleného terminálu. V opačném případě je pouze odeslán řetězec: `Login failed, incorrect username or password` a spojení je serverem ukončeno. Průběh typického postupu autorizace ke vzdálenému systému pomocí protokolu MT je uveden ve schématu na obrázku na následující straně.



Obrázek 7.4: Průběh autorizace ke vzdálenému terminálu protokolem MT

7.4.2 MikroTik WinBox

Protokol MikroTik WinBox (MWB) představuje opět nedokumentovaný proprietární protokol, který nemá žádnou hlavičku. Jedná se pouze o tok řídicích příkazů z/do zařízení s OS MikroTik RouterOS. Na počátku komunikace je nutné autorizovat přístup k zařízení. Princip autorizace je obdobný jako u protokolu MT. Nejprve je na základě výzvy klientem o náhodný řetězec zaslán serverem 16znakový náhodný řetězec, na jehož základě je vypočítán pomocí hašovací funkce MD5 autorizační řetězec. Ten je společně s náhodným řetězcem, autorizačním řetězcem a uživatelským jménem zaslán zpět do zařízení s OS MikroTik RouterOS, vůči kterému je autorizace prováděna.

7.4.3 MikroTik WinBox Secure

Data protokolu MikroTik WinBox Secure (MWBS) pravděpodobně vzniknou zašifrováním dat z protokolu MikroTik WinBox. Protokol MikroTik WinBox Secure tedy pravděpodobně nepředstavuje separátní protokol, ale jsou to pouze zašifrovaná data protokolu MikroTik WinBox pravděpodobně protokolem TLS. Vztah mezi protokoly MWB a MWBS lze předpokládat obdobný jako je např. u standardizovaných HTTP a HTTPS protokolů.

8 KONCEPCE REALIZOVANÉHO ŘEŠENÍ

V první variantě řešení bylo předpokládáno použití proprietární sestavy protokolů MTP-MWB, ale protokol MWB se v průběhu analýzy ukázal jako příliš komplikovaný a náročný na systémové prostředky. Jeho podstatnou výhodou ovšem je, že pomocí jednoho spojení (session) lze paralelně provádět neomezený počet operací. Zásadním problémem je, že protokol pracuje přímo s binárními daty a jeho analýza je proto velmi komplikovaná. Také objem dat, které je nutno analyzovat, je značný. Nejzávažnější nevýhodou ovšem je, že pro každou realizovanou funkci by bylo nutné provést samostatnou analýzu komunikace, z čehož plyne, že každému přidaní nové funkce do knihovny automatické vzdálené správy by předcházela analýza potřebné komunikace, ve které se vyskytuje použití dané funkce. Neexistuje možnost vytvoření jednotného principu, pomocí kterého se přenáší všechny příkazy. Kvůli těmto zásadním nevýhodám bylo od použití řešení využívajícího proprietární sestavy protokolů MTP-MWB upuštěno.

V nově navrženém řešení je využívána proprietární sestava protokolů MTP-MT, u které se v průběhu analýzy použitých proprietárních protokolů v OS MikroTik RouterOS podařilo vytvořit kompletní popis komunikace, kterou tato sestava protokolů využívá. Tento popis zahrnuje jak význam jednotlivých oktetů v ethernetovém rámci, tak i stavové chování obou analyzovaných protokolů. Podstatnou výhodou je, že bylo možné vytvořit jednotný popis principu funkce obou analyzovaných protokolů, pomocí kterého lze přenést do spravovaného zařízení s OS MikroTik RouterOS libovolný příkaz, aniž by bylo nutné realizovat jakoukoliv doplňující analýzu pro daný konkrétní příkaz. Toto je zásadní výhoda oproti řešení využívající proprietární sestavu protokolů MTP-MWB. Taktéž je předpokládaná bezproblémová kompatibilita řešení s různými verzemi OS MikroTik RouterOS, protože klient pro tento protokol pro platformu Win32 se od roku 2004 nezměnil a bylo prakticky ověřeno, že pracuje jak s dnes již historickou verzí 3 OS MikroTik RouterOS, tak i s nejnovější verzí 5. Proto lze předpokládat bezproblémovou funkčnost navrženého řešení i s budoucími verzemi OS MikroTik RouterOS.

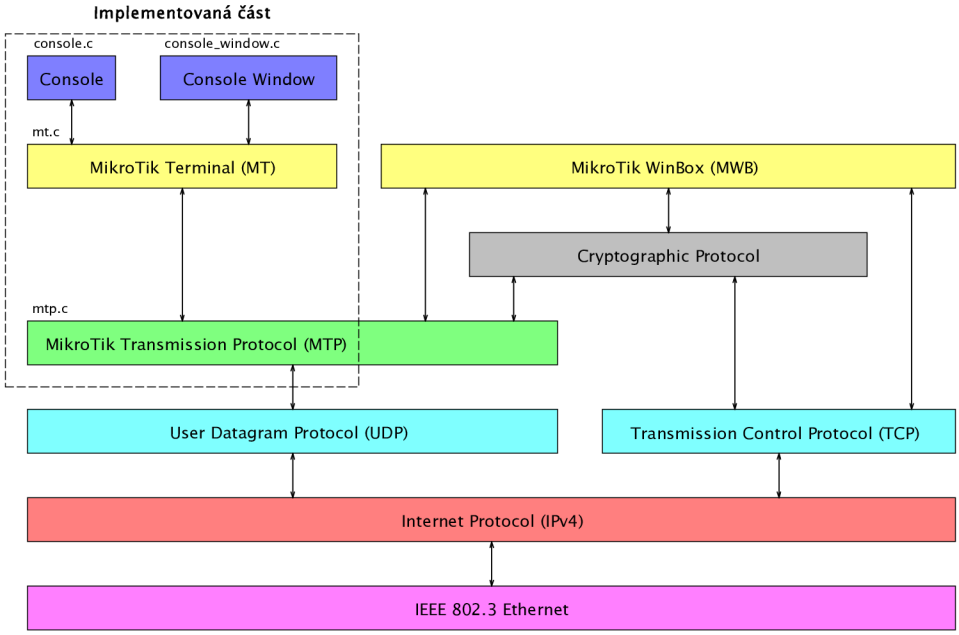
Automatická vzdálená správa je z vnějšího pohledu realizována prostřednictvím příkazového řádku vzdáleného systému. Data, která by byla využita pro aktualizaci terminálového okna, jsou analyzována na výsledky zadaných příkazů.

Výhodou zvoleného řešení využívajícího protokol MTP je adresace zařízení na základě MAC adresy jejich síťového rozhraní, která je nezávislá na aktuální konfiguraci IP protokolu. Z toho ovšem plyne i omezení mající vliv na použitelnost komunikace založené na tomto protokolu. Vzhledem k tomu, že je využíváno broadcastové vysílání pomocí protokolu UDP, je možné komunikovat pomocí tohoto protokolu pouze v rámci jednoho segmentu sítě, což ale není významná nevýhoda, protože mobilní platforma bude pracovat v síti, která bude tvořena pouze jedním segmentem.

9 POPIS IMPLEMENTOVANÉHO ŘEŠENÍ

Jednotlivé moduly realizované knihovnu řešící automatickou vzdálenou správu zařízení s OS MikroTik RouterOS jsou podle zadání napsány v jazyce C. Zvolená implementace podle doporučení vedoucího práce nevyužívá dynamickou alokaci paměti, ale využívá jen statickou alokaci paměti. Z toho plyne omezení, že některé omezující parametry je nutné nastavit pomocí maker před kompilací knihovny. Zejména se jedná o maximální počet současně otevřených spojení (session). Podle požadavků vedoucího diplomové práce jsou všechny funkce realizovány jako neblokovací, tedy pro návrat z jejich volání není vyžadováno dokončení započaté operace. Funkce jsou tedy uzpůsobeny pro provoz v supersmyčce. Jádro knihovny nevyužívá žádné funkce externích knihoven, takže by mělo být bez větších problémů přenositelné i na jinou platformu. Ohled na přenositelnost vytvořeného kódu na embedded systém mobilní platformy byl jedním z požadavků zadání. Funkčnost celého řešení byla ověřena na platformě Win32, konkrétně na operačním systému Microsoft Windows XP Professional SP3.

Celé řešení je realizováno pomocí několika samostatných modulů, respektujících rozdělení celé komunikace do samostatných vrstev, kdy každá vrstva řeší izolovaně od ostatních vrstev vybraný úkol. Každá z vrstev může přímo komunikovat jen s nadřazenou a podřízenou vrstvou. S ostatními vrstvami může daná vrstva komunikovat pouze prostřednictvím nadřazené nebo podřazené vrstvy. Klíčové moduly realizované knihovny (mtp.c, mt.c, console.c a console_window.c) a jejich návaznost na ostatní moduly je ukázána na následujícím schématu:



Obrázek 9.1: Klíčové moduly realizované knihovny

Zvolená implementace umožňuje, aby každá z realizovaných vrstev mohla mít v jednom okamžiku otevřený libovolný počet aktivních komunikací. Tento počet je pouze omezen alokovaným počtem tzv. slotů. Do jednoho slotu v dané vrstvě lze uložit stavové informace o jedné probíhající komunikaci na dané vrstvě. Počet těchto slotů na každé z implementovaných vrstev se volí hodnotou makra MTP_SESSIONS_MAX v hlavičkovém souboru mtp.h.

Všechny implementované funkce vrací buď hodnotu příslušející výsledku požadované operace nebo v případě chyby vrací hodnotu -1. V případě logické hodnoty výsledku požadované operace je v kladném případě navrácena hodnota 0, jinak je navrácena hodnota -1 (i v případě chyby této operace).

9.1 Modul mtp.c

Modul mtp.c obsahuje implementaci analyzovaného protokolu MTP. Tento modul zajišťuje obsluhu všech otevřených spojení (session) s danými servery. Úkolem tohoto modulu je vytvořit, udržovat a ukončit spojení se zadanými servery. Z pohledu vyšší vrstvy komunikačního modelu zajistí tento protokol vytvoření komunikačního kanálu, který garantuje spolehlivé doručování dat a jejich doručování ve správném pořadí. Spojení protokolů UDP a MTP má tedy obdobné vlastnosti jako transportní protokol TCP. Každé aktivní spojení má přidělen svůj vyhrazený slot, ve kterém má uloženy své stavové proměnné.

Funkce tvořící uživatelské rozhraní

```
int InitMTP();
```

Funkce provede výchozí inicializaci alokovaných slotů pro protokol MTP.

```
int OpenNewSession(unsigned char *climac,
                   unsigned char *srvmac,
                   unsigned int   sessid,
                   char           slot,
                   struct ring_buffer_t *ring_buffer);
```

Funkce otevře nové spojení (session) ve vybraném slotu.

`climac`	ukazatel na první oktet MAC adresy klienta
`srvmac`	ukazatel na první oktet MAC adresy serveru
`sessid`	požadovaný jednoznačný identifikátor nového spojení obsahující i kód neseného protokolu použitého ve vyšší vrstvě
`slot`	číslo slotu, ve kterém má být nové spojení vytvořeno
`ring_buffer`	ukazatel na kruhový zásobník (pokud je použit, jinak je NULL)

```
int WriteDataToSession(char *data, int len, char slot);
```
Funkce sloužící pro zápis dat do aktivního spojení v zadaném slotu.

`data`	ukazatel na pole dat, která mají být do spojení zapsána
`len`	délka pole dat
`slot`	číslo slotu určující spojení, do něhož mají být data zapsána

```
int ProcessMTPSegment(char *mtp_segment, int len);
```
Funkce pro zpracování přijatého MTP segmentu.

`mtp_segment`	ukazatel na první oktet přijatého MTP segmentu
`len`	délka přijatého MTP segmentu

```
char SessionInactive(char slot);
```
Funkce testuje, zda-li je zvolený slot volný (spojení v daném slotu je neaktivní).

`slot`	číslo slotu, ve kterém se má ověřovat stav spojení

```
char SessionReady(char slot);
```
Funkce testuje, zda-li je dané spojení v zadaném slotu připraveno odesílat data.

`slot`	číslo slotu, ve kterém se má ověřovat stav spojení

```
char SessionActive(char slot);
```
Funkce testuje, zda-li je spojení ve zvoleném slotu v aktivním stavu. Aktivní stav spojení znamená, že v budoucnu bude možno pomocí tohoto spojení přijímat a odesílat data. Může nastat situace, kdy vybraný slot nemusí být volný a přesto v budoucnosti již nebude možné přes zvolené spojení odesílat ani přijímat data (např. při uzavírání spojení ve vybraném slotu).

`slot`	číslo slotu, ve kterém se má ověřovat stav spojení

```
int StateAutomataMTP();
```
Funkce aktualizuje stav stavového automatu použitého pro každý slot. Funkci je tedy nutné volat při každém cyklu supersmyčky.

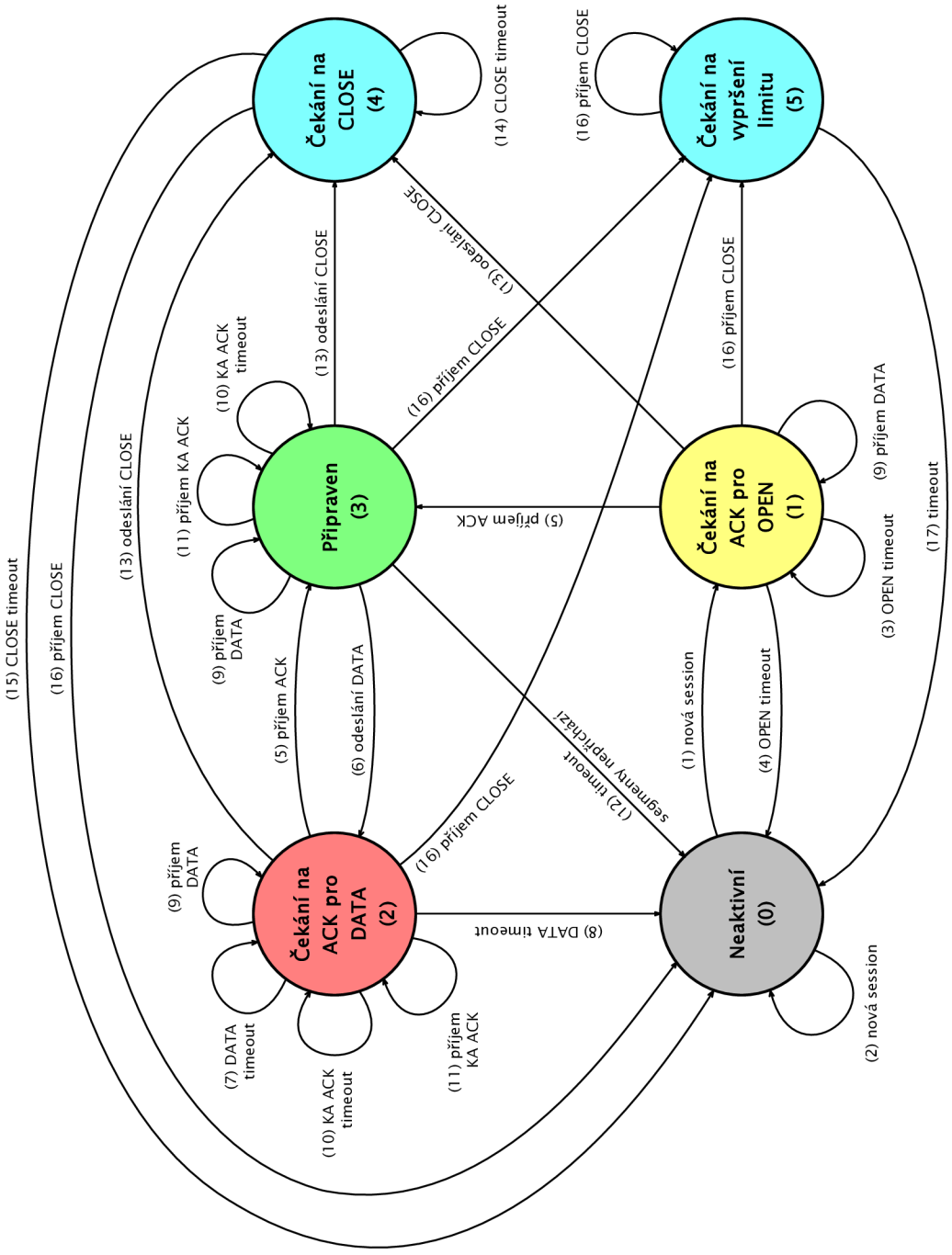
```
int CloseSession(char slot);
```
Funkce uzavře aktivní spojení ve vybraném slotu.

`slot`	číslo slotu, ve kterém má být spojení ukončeno

```
int LogSessionsInfo();
```
Funkce provede zaprotokolování informací o stavu spojení v jednotlivých slotech.

Nové spojení lze otevřít voláním funkce `OpenNewSession` s akceptovatelnými parametry, jinak nebude spojení otevřeno. Po akceptování tohoto požadavku je odeslán MTP segment typu OPEN a je vyčkáváno na příjem potvrzujícího MTP segmentu typu ACK, aby bylo možné přejít ze stavu (1) *Čekání na ACK pro OPEN* do stavu (3) *Připraven*. Po přechodu do stavu (3) *Připraven* je umožněno odesílat data. Po odeslání dat serveru je proveden přechod do stavu (2) *Čekání na ACK pro DATA*. V tomto stavu se čeká na příjem potvrzujícího platného MTP segmentu typu ACK. Ze všech aktivních stavů (1, 2, 3) je možné spojení ukončit jak ze strany klienta tak i ze strany serveru.

Pokud je v těchto stavech přijat MTP segment typu CLOSE, je tento MTP segment potvrzen MTP segmentem typu CLOSE a je proveden přechod do stavu (5) *Čekání na vypršení limitu*. Ten je nutný pro potvrzování případných dalších přijatých MTP segmentů typu CLOSE, v případě, že se odeslaný MTP segment typu CLOSE ztratil. Při ukončení spojení ze strany klienta je odeslán MTP segment typu CLOSE a je provedeno čekání na jeho potvrzení. Schéma stavového automatu jednoho slotu, podle něhož je provedena implementace protokolu MTP, je uvedeno na následujícím obrázku:



Obrázek 9.2: Stavový diagram implementace MTP protokolu

9.2 Modul mt.c

Modul mt.c implementuje správu terminálového spojení. Zejména se jedná o provedení autorizačního procesu a o informování serveru o změně velikosti terminálového okna. Po provedení autorizačního procesu modul předává data zpracovaná modulem mtp.c modulům ve vyšších vrstvách (console.c nebo console_window.c) nebo tato data zapisuje do kruhového zásobníku.

Funkce tvořící uživatelské rozhraní

```
int InitMT();
```

Funkce provede výchozí inicializaci alokovaných slotů pro protokol MT.

```
int OpenNewTerminal(unsigned char *climac,
                    unsigned char *srvmac,
                    int   sessnm,
                    char *username,
                    char *password,
                    char *term_mode,
                    int   width,
                    int   height,
                    char  slot,
                    char  hi_layer,
                    struct ring_buffer_t *ring_buffer);
```

Funkce provede otevření nového terminálu ze zadanými parametry.

`climac`	ukazatel na první oktet MAC adresy klienta
`srvmac`	ukazatel na první oktet MAC adresy serveru
`sessnm`	číslo spojení (povolený rozsah 0x0000 až 0xFFFF)
`username`	ukazatel na první znak uživatelského jména (nulou zakončený řetězec)
`password`	ukazatel na první znak uživatelského hesla (nulou zakončený řetězec)
`term_mode`	ukazatel na první znak řetězce módu terminálu (nulou zakončený řetězec)
`width`	počet znaků na řádek okna terminálu
`height`	počet řádků okna terminálu
`slot`	číslo slotu, ve kterém má být otevřen nový terminál (a současně i spojení)
`hi_layer`	číslo protokolu vyšší vrstvy (1 – Console, 2 – Console Window)
`ring_buffer`	ukazatel na kruhový zásobník (pokud je použit, jinak je NULL)

```
int WriteTextToTerminal(char *text, int len, char slot);
```

Funkce zapíše řetězec znaků do terminálu v zadaném slotu.

`text`	ukazatel na řetězec znaků (může obsahovat i nulový znak)
`len`	délka řetězce znaků
`slot`	číslo slotu definující terminál, do kterého má být řetězec zapsán

```
int ProcessMTData(char *mt_data, int len, char slot);
```
Funkce provede zpracování dat protokolu MT (uživatelská data MTP segmentu).

`mt_data`	ukazatel na první oktet dat protokolu MT v ethernetovém rámci
`len`	délka dat určených ke zpracování protokolem MT
`slot`	číslo slotu identifikující terminál, kterému data přísluší

```
char TerminalInactive(char slot);
```
Funkce ověří, zda-li je zadaný slot volný (terminál v daném slotu je neaktivní).

`slot`	číslo slotu, ve kterém má být ověřen stav terminálu

```
char TerminalReady(char slot);
```
Funkce ověří, zda-li je terminál v zadaném slotu připraven odesílat data.

`slot`	číslo slotu, ve kterém má být ověřen stav terminálu

```
char TerminalActive(char slot);
```
Funkce ověří zda-li je terminál v zadaném slotu v aktivním stavu (význam funkce je obdobný jako u funkce `SessionActive`).

`slot`	číslo slotu, ve kterém má být ověřen stav terminálu

```
int SetTerminalSize(int width, int height, char slot);
```
Funkce odešle serveru informaci o změně velikosti terminálového okna.

`width`	nový počet znaků na řádek terminálového okna
`height`	nový počet řádků terminálového okna
`slot`	číslo slotu definující terminál, kterého se změna týká

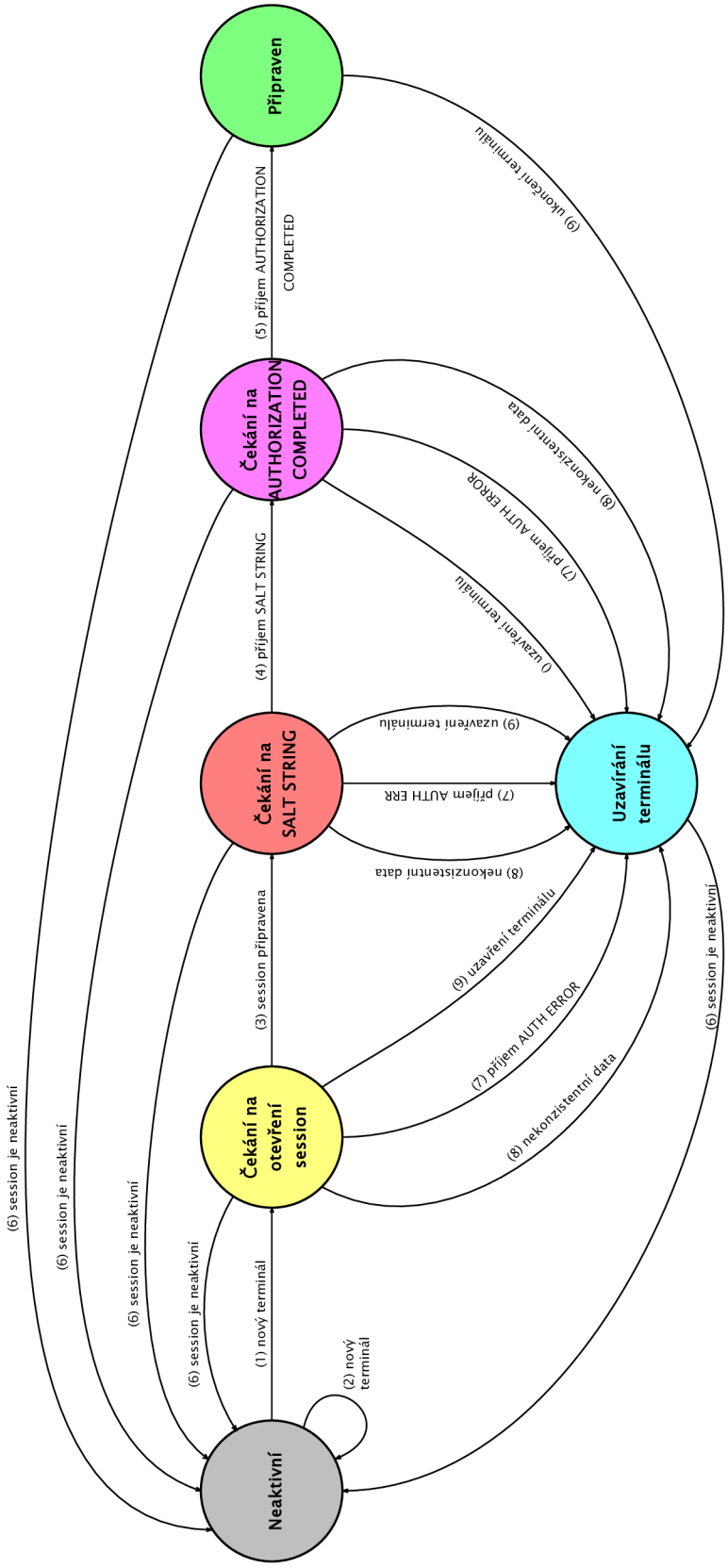
```
int StateAutomataMT();
```
Funkce aktualizuje stav stavových automatů všech slotů. Funkci je nutné volat při každém cyklu supersmyčky.

```
int CloseTerminal(char slot);
```
Funkce provede uzavření aktivního terminálu v zadaném slotu.

`slot`	číslo slotu, v němž má být terminál uzavřen

Ve výchozím stavu je terminál v příslušném slotu neaktivní. Pro aktivaci je nutné zavolat funkci `OpenNewTerminal`. V případě, že nový terminál není možné se zadanými parametry otevřít, přechod z neaktivního stavu není uskutečněn. V kladném případě se přejde do stavu, ve kterém se čeká na otevření nového spojení. Po otevření nového spojení je odeslána serveru informace o započetí autorizačního procesu. Klient čeká na příjem náhodného řetězce. Po přijetí náhodného řetězce vypočítá autorizační řetězec a spolu s dalšími identifikátory jej zašle serveru. Klient nyní čeká na příjem informace o ukončení autorizačního procesu. Pro jeho přijetí je proveden přechod do stavu *Připraven*. V tuto chvíli je možné pomocí funkce `WriteDataToTerminal` zapsat řetězec do příkazové řádky vzdáleného systému. V případě příjmu informace o neplatnosti autorizačního procesu v průběhu autorizace, je proveden přechod do stavu *Uzavírání terminálu*. Při výpadku spojení se serverem je terminál okamžitě ukončen.

Stavový diagram na jehož základě je provedena implementace protokolu MT:



Obrázek 9.3: Stavový diagram implementace MT protokolu

9.3 Modul console.c

Tento modul realizuje samotnou automatickou vzdálenou správu zařízení s OS MikroTik RouterOS. Modul provádí automatické zpracování textových řetězců (konzolových dat) přijímaných a zasílaných ze/do spravovaného zařízení s OS MikroTik RouterOS. V tomto modulu je realizováno např. měření datových toků na jednotlivých portech spravovaného zařízení. Modul automaticky odešle potřebný příkaz a vyhledává v přijatých datech specifické textové řetězce, které obsahují naměřená data a ty převádí do číselné reprezentace. V tomto modulu je rovněž řešena detekce volné příkazové řádky (připravenost vzdáleného systému k vložení příkazu), ukončení provádění příkazu a jiné. Zjednodušeně lze říci, že tento modul simuluje činnost osoby provádějící správu zařízení s OS MikroTik RouterOS.

Funkce tvořící uživatelské rozhraní

```
int InitConsoles();
```

Funkce provede výchozí inicializaci alokovaných slotů pro modul console.c

```
int OpenNewConsole(unsigned char *climac,
                   unsigned char *srvmac,
                   int   sessnm,
                   char *username,
                   char *password,
                   char  slot);
```

Funkce zajistí otevření nové konzole pro zadávání příkazů.

`climac`	ukazatel na první oktet MAC adresy klienta
`srvmac`	ukazatel na první oktet MAC adresy serveru
`sessnm`	identifikační číslo nového spojení, které bude pro tuto konzoli vytvořeno
`username`	ukazatel na nulou zakončený řetězec obsahující uživatelské jméno
`password`	ukazatel na nulou zakončený řetězec obsahující uživatelské heslo
`slot`	číslo slotu, ve kterém má být nové konzolové okno (a současně terminál i spojení) vytvořeno

```
int ProcessConsoleData(char *c_data, int len, char slot);
```

Funkce zajistí zpracování konzolových dat, které jsou výsledkem zpracování dat v modulu mt.c.

`c_data`	ukazatel na data určená ke zpracování v modulu console.c
`len`	délka dat určených ke zpracování
`slot`	číslo slotu, kterému vstupní data přísluší

```
int StartMonitorTraffic(char *interf, char slot);
```

Funkce zahájí měření datových toků na zadaném rozhraní.

`interf`	ukazatel na nulou zakončený řetězec obsahující název měřeného rozhraní
`slot`	číslo slotu určující aktivní konzoli, ve které má být měření spuštěno

```
int EndMonitorTraffic(char slot);
```
Funkce ukončuje spuštěné měření datových toků.

`slot` číslo slotu, určující konzoli, ve které je měření datových toků spuštěno

```
char GetROSversion(char slot);
```
Funkce navrátí číslo hlavní verze OS MikroTik RouterOS.

`slot` číslo slotu, určující konzoli, ze které má být údaj získán

```
char GetROSsubversion(char slot);
```
Funkce navrátí číslo podverze OS MikroTik RouterOS.

`slot` číslo slotu, určující konzoli, ze které má být údaj získán

```
int GetRX_pps(char slot);
```
Funkce navrací počet přijatých paketů za sekundu na dříve zadaném rozhraní.

```
int GetRX_bps(char slot);
```
Funkce navrací počet přijatých bajtů za sekundu na dříve zadaném rozhraní.

```
int GetTX_pps(char slot);
```
Funkce navrací počet odeslaných paketů za sekundu na dříve zadaném rozhraní.

```
int GetTX_bps(char slot);
```
Funkce navrací počet odeslaných bajtů za sekundu na dříve zadaném rozhraní.

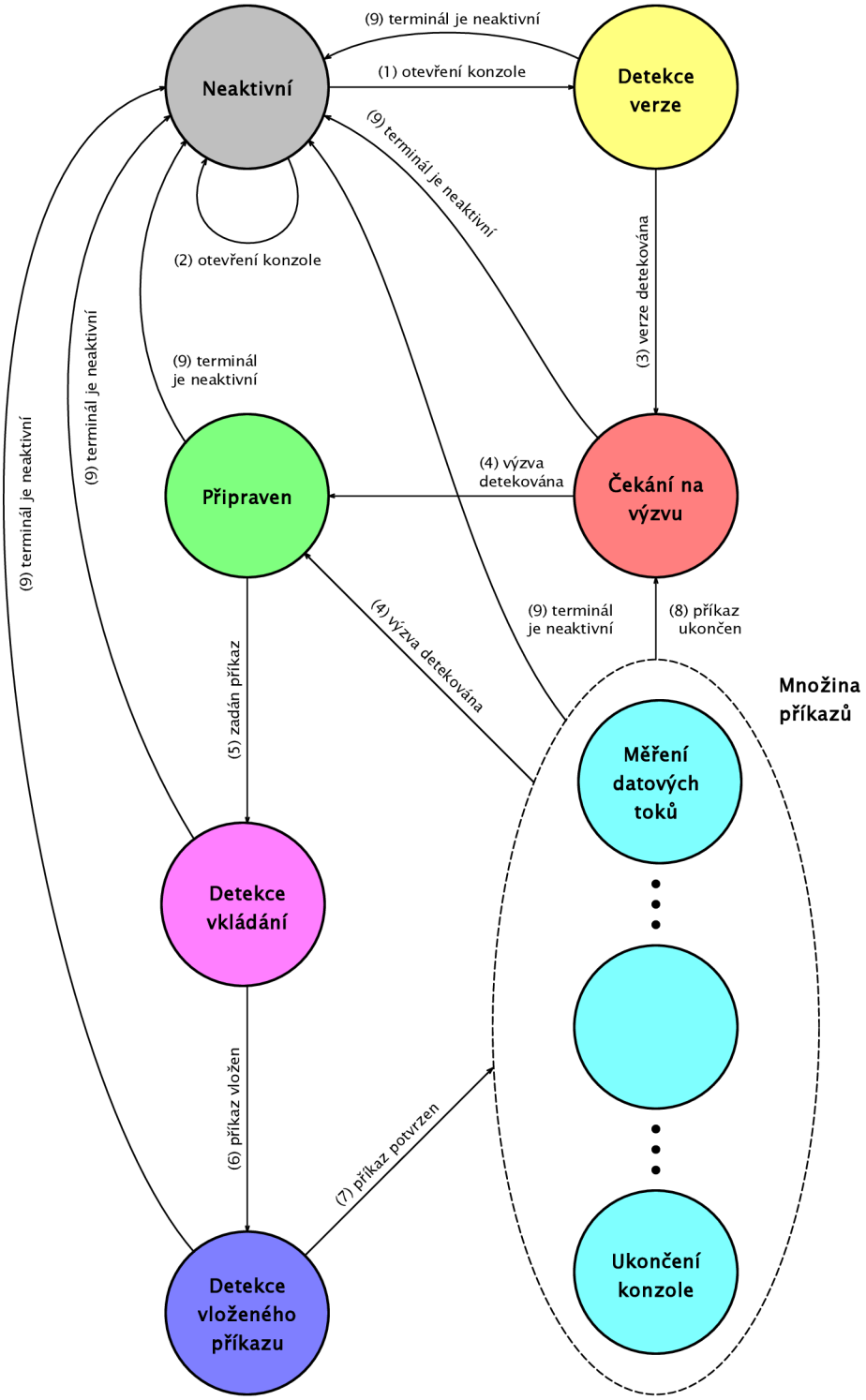
```
int StateAutomataConsoles();
```
Funkce provede aktualizaci stavu všech stavových automatů příslušejících jednotlivým slotům.

```
int CloseConsole(char slot);
```
Funkce uzavře aktivní konzoli v zadaném slotu.

`slot` číslo slotu, ve kterém má být konzole ukončena

Po akceptování požadavku na otevření nové konzole pomocí funkce `OpenNewConsole` je proveden přechod do stavu *Detekce verze*. Z tohoto stavu je po přijetí příslušného řetězce znaků proveden přechod do stavu *Čekání na výzvu*, ve kterém se čeká na detekování volné příkazové řádky. Po její detekci je proveden přechod do stavu *Připraven*. V tomto stavu je možné vložit do konzole požadovaný příkaz. Po provedení požadavku na vložení příkazu je do konzole požadovaný příkaz vložen a následně je ověřeno, že vložení proběhlo úspěšně (ve stavu *Detekce vkládání*) a příkaz byl akceptován (ve stavu *Detekce vloženého příkazu*). Po tomto úvodním ověření správnosti vložení zadaného příkazu je proveden přechod do stavu příslušejícímu danému příkazu. Z tohoto stavu je možné přejít po předání pokynu k ukončení příkazu do stavu, ve kterém se čeká na přítomnost volné příkazové řádky a po její detekci je možné přejít zpět do stavu *Připraven*. V případě detekování volné příkazové řádky v průběhu vykonávání příkazu (signalizuje samovolné ukončení prováděného příkazu) je proveden přechod přímo do stavu *Připraven*. V případě výpadku terminálu je konzole ukončena.

Stavový diagram, podle něhož je implementováno zpracování dat v modulu console.c:



Obrázek 9.4: Stavový diagram provádění operací na vzdáleném systému

9.4 Modul console_window.c

Protože samotný protokol MT umožňuje zprostředkovat příkazovou řádku vzdáleného systému, je realizován i modul console_window.c, který zpracovává obsluhu terminálového okna. Tento modul lze tedy využít pro realizaci klienta umožňujícího připojení ke vzdálenému systému protokolem MT. Samotný modul především analyzuje vstupní data na výskyt tzv. escape sekvencí a ostatní data vypisuje do terminálového okna. Na základě detekovaného typu escape sekvence je vykonána operace příslušející této escape sekvenci (přemístění kurzoru, změna barvy textu nebo pozadí, ...).

Funkce tvořící uživatelské rozhraní

```
int InitConsoleWindows();
```

Funkce inicializuje do výchozího stavu alokované sloty pro modul console_window.c

```
int OpenNewConsoleWindow(unsigned char *climac,
                         unsigned char *srvmac,
                         int    sessnm,
                         char  *username,
                         char  *password,
                         char   slot,
                         HANDLE hConsoleOutput);
```

Funkce zajistí otevření nového konzolového okna se zadanými parametry

- `climac` ukazatel na první oktet MAC adresy klienta
- `srvmac` ukazatel na první oktet MAC adresy serveru
- `sessnm` identifikační číslo nového spojení, které bude pro toto okno vytvořeno
- `username` ukazatel na nulou zakončený řetězec obsahující uživatelské jméno
- `password` ukazatel na nulou zakončený řetězec obsahující uživatelské heslo
- `slot` číslo slotu ve kterém má být nové konzolové okno (a současně terminál i spojení) vytvořeno
- `hConsoleOutput` handle na předem alokované konzolové okno v systému Win32

```
int WriteTextToConsoleWindow(char *text, int len, char slot);
```

Funkce zapíše textový řetězec do zvoleného konzolového okna.

- `text` ukazatel na první znak řetězce, který má být do konzolového okna zapsán
- `len` délka zapisovaného řetězce
- `slot` číslo slotu, identifikující dané konzolové okno

```
int ProcessConsoleWindowData(char *c_data, int len, char slot);
```

Funkce zpracovává konzolová data, která jsou výsledkem zpracování dat v modulu mt.c

- `c_data` ukazatel na řetězec dat předávaný ke zpracování
- `len` délka řetězce dat
- `slot` číslo slotu, určující kterému konzolovému oknu data přísluší

```
int StateAutomataConsoleWindows();
```
Funkce provede aktualizaci stavu všech stavových automatů jednotlivých slotů.

```
int ConvertKeyCode(int key_code, char **sequence);
```
Funkce konvertuje kód klávesy na sekvenci dat, kterou je třeba odeslat vzdálenému systému, aby provedl operaci příslušející dané klávese.

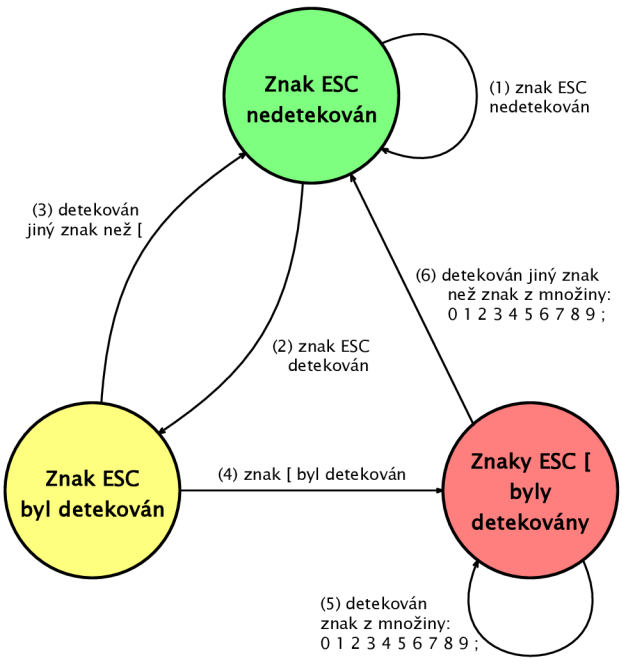
`key_code` kód klávesy, který je třeba převést na sekvenci

`sequence` ukazatel na místo, kam má být uložen ukazatel na příslušnou sekvenci dat, odpovídající dané klávese

```
int CloseConsoleWindow(char slot);
```
Funkce uzavře aktivní konzolové okno v zadaném slotu

`slot` číslo slotu, ve kterém má být konzolové okno uzavřeno

Zpracování přijatých dat se řídí aktuálním stavem stavového automatu vyhodnocujícího význam právě přijatých dat. Pokud není detekován znak `ESC` (0x1B) je příslušný znak vypsán do fyzického konzolového okna. Pokud je detekován znak `ESC` je proveden přechod do stavu *Znak ESC byl detekován* a tento znak ani následující přijatý znak, který rozhoduje o typu sekvence není vypisován. Pokud se nejedná o znak `[`, je přijatá escape sekvence tvořena pouze dvouznakovou sekvencí. Po provedení příslušného příkazu je proveden přechod do výchozího stavu, kdy jsou přijaté znaky vypisovány do fyzického konzolového okna. V případě přijetí znaku `[` je proveden přechod do stavu *Znaky ESC [byly detekovány* a opuštění tohoto stavu je možné až po přijetí jiného znaku než číslic a středníku. Schéma stavového automatu je na následujícím obrázku:



Obrázek 9.5: Stavový diagram rozkladu konzolových dat

9.5 Modul network_interface.c

Tento modul zajišťuje předání dat síťovému rozhraní a jeho implementace je tedy závislá na použité platformě. Provedená realizace tohoto modulu je určena pro platformu Win32. Modul využívá prostředky soketového API platformy Win32. Modul vytvoří jeden standardní datagramový soket pro protokol UDP, pomocí kterého odesílá a přijímá data pro všechna spojení (session).

Funkce tvořící uživatelské rozhraní

```
int InitInterface(int udp_port);
```

Funkce zajistí inicializaci síťového rozhraní (založení datagramového soketu).

`udp_port`	UDP port na kterém naslouchají MTP servery

```
int GetMACaddress(unsigned char *MACaddress, char adapter);
```

Funkce navrací MAC adresu zvoleného síťového adaptéru PC.

`MACaddress`	ukazatel na alokovaný prostor 6 bajtů pro uložení MAC adresy
`adapter`	pořadové číslo síťového adaptéru

```
int SendMTPSegment(char *mtp_segment, int len);
```

Funkce pro odeslání MTP segmentu.

`mtp_segment`	ukazatel na první oktet MTP segmentu
`len`	počet oktetů daného MTP segmentu

```
int ReceiveMTPSegment(char *mtp_segment, int len);
```

Funkce zajistí vyzvednutí MTP segmentu ze zásobníku TCP/IP stacku.

`mtp_segment`	ukazatel na první oktet alokovaného pole pro uložení přijatého MTP segmentu
`len`	velikost alokované pole (max. velikost MTP segmentu)

```
int CloseInterface();
```

Funkce provede uzavření síťového rozhraní (uzavřené datagramového soketu).

9.6 Modul ring_buffer.c

Modul ring_buffer.c implementuje funkce pro práci s kruhovým zásobníkem, který může být použit pro uložení zpracovaných dat některým z modulů (mtp.c, mt.c). Toho lze využít tak, že data nemusí být zpracována při příjmu všemi vrstvami, ale výstupní data, která by byla předána vyšší vrstvě (např. MT od MTP) můžou být zapsána do kruhového zásobníku. Tento modul implementuje základní operace jako je inicializace kruhového zásobníku, zápis a čtení dat do/z kruhového zásobníku.

Funkce tvořící uživatelské rozhraní

```
int RingBufferInit(struct ring_buffer_t *ring_buffer);
```

Funkce provede inicializaci zadaného kruhového zásobníku

`ring_buffer`	ukazatel na strukturu kruhového zásobníku

```
int ReadFromRingBuffer(struct ring_buffer_t *ring_buffer,
                       char *data,
                       int   len);
```

Funkce načte data z kruhového zásobníku do zadaného pole.

`ring_buffer`	ukazatel na strukturu kruhového zásobníku
`data`	ukazatel na pole, do kterého mají být data uložena
`len`	délka alokovaného pole (max. počet přečtených dat)

```
int WriteToRingBuffer(struct ring_buffer_t *ring_buffer,
                      char *data,
                      int   len);
```

Funkce provede zápis dat do zvoleného kruhového zásobníku.

`ring_buffer`	ukazatel na strukturu kruhového zásobníku
`data`	ukazatel na pole dat, která mají být do kruhového zásobníku zapsána
`len`	délka pole dat

9.7 Modul timing.c

Jedná se o modul zpracovávající hodnotu běžícího čítače s frekvencí přibližně 3,58 MHz, jehož údaj je poskytován operačními systémy Microsoft Windows. Tento modul je tedy závislý na použité platformě (Win32). Tento údaj je pro další využití přepočítán na údaj s krokem 1 ms, což je dostatečný krok změny pro měření potřebných událostí. Ostatními moduly je tento modul využíván jako zdroj synchronizačního času, zejména pro měření vypršení časových limitů, které se pohybují v řádu desítek ms.

Funkce tvořící uživatelské rozhraní

```
int InitTiming();
```

Funkce pro inicializaci modulu volně běžícího čítače.

```
unsigned int MillisecondsCounter();
```

Funkce navrací aktuální počet milisekund od startu PC.

9.8 Modul logging.c

Modul logging.c zajišťuje podporu protokolování činnosti všech ostatních modulů obsažených ve vyvinuté komunikační knihovně. Jedná se zejména o vytvoření několika samostatných protokolovacích souborů a jednoho souhrnného protokolovacího souboru, zápis zpráv do jednotlivých protokolovacích souborů, výpis bloku paměti do protokolovacího souboru a uzavření protokolovacích souborů.

Soubor	Modul	Popis
log.log	všechny moduly	souhrnný protokol pro všechny moduly
log0.log	main.c	výchozí protokol
log1.log	network_interface.c	výchozí protokol
log2.log	network_interface.c	pouze příjem/odeslání MTP segmentu
log3.log	mtp.c	výchozí protokol
log4.log	mtp.c	pouze zpracování přijatých dat
log5.log	mt.c	výchozí protokol
log6.log	mt.c	pouze zpracování přijatých dat
log7.log	console.c	výchozí protokol
log8.log	console.c	pouze zpracování přijatých dat
log9.log	timing.c	výchozí protokol
log10.log	ring_buffer.c	výchozí protokol
log11.log	console_window.c	pouze zpracování přijatých dat

Tabulka 9.1: Popis protokolovacích souborů

Funkce tvořící uživatelské rozhraní

```
int CreateLogs();
```
Funkce založí protokolovací soubory podle definic v hlavičkovém souboru logging.h.

```
int WriteHeaderToLogs();
```
Funkce zapíše hlavičky do všech protokolovacích souborů.

```
int LogMessage(char *message, int log, int level);
```
Funkce pro zaprotokolování zprávy.

- `message` ukazatel na nulou zakončený řetězec obsahující zprávu
- `log` pořadové číslo protokolu, do kterého má být zpráva zapsána
- `level` úroveň důležitosti zprávy (chyba, varování, informace)

```
int LogMemory(char *memory, int len, int log, int level);
```
Funkce pro zaprotokolování obsahu paměti ve formátu HEX s ASCII vyjádřením.

- `memory` ukazatel na první bajt paměťového místa
- `len` délka bloku dat v paměti
- `log` pořadové číslo protokolu, do kterého má být obsah paměti zapsán
- `level` důležitost protokolování obsahu paměti (chyba, varování, informace)

```
int CloseLogs();
```
Funkce uzavře protokolovací soubory.

9.9 Modul md5.c

Jedná se o modul, který realizuje výpočet MD5 haše ze zadaného řetězce. Tato hašovací funkce je potřebná pro výpočet autorizačního řetězce v MT protokolu. Soubory md5.h a md5.c jsou převzaty z [10].

9.10 Modul main.c

Jedná se o hlavní modul, na jehož základě se kompiluje spustitelná aplikace pro platformu Win32, která ukazuje použití celé knihovny pro automatickou vzdálenou správu zařízení s OS MikroTik RouterOS. Parametry připojení se volí v souboru settings.ini. Aplikace ukazuje funkčnost všech modulů na úloze měření datových toků na vybraném rozhraní zařízení s OS MikroTik RouterOS a na současně otevřeném oknu s příkazovou řádkou vzdáleného systému. Ukázka okna této aplikace je na následujícím obrázku:

Obrázek 9.6: Ukázka konzolového okna realizované aplikace

10 ZÁVĚR

V uvedené práci je zdokumentováno řešení problému automatické vzdálené správy zařízení s OS MikroTik RouterOS, které pracuje jako bezdrátový přístupový bod, který je součástí mobilní platformy vyvíjené na UAMT FEKT VUT. Práce bere v úvahu omezení, která vycházejí z použité mobilní platformy a navrhuje jednu konkrétní variantu řešení problému, která respektuje tato omezení a vyhovuje všem kladeným požadavkům na výsledné řešení.

V úvodu práce jsou rozebrány možnosti správy zařízení, na kterém je korektně spuštěn OS MikroTik RouterOS. Krátce jsou popsány dostupné aplikace pro správu, využívající standardizovaných komunikačních protokolů. Dále jsou popsány možnosti a základní princip výrobcem dodávaných specializovaných aplikací (WinBox a Terminal) pro správu zařízení s tímto operačním systémem. Tyto aplikace ale využívají vlastních proprietárních komunikačních protokolů, jejichž dokumentace není v současnosti veřejně dostupná.

Velká část této práce se zabývá analýzou a popisem komunikace, kterou využívají právě zmíněné aplikace pro správu systému dodané výrobcem, tedy komunikací založených na proprietárních protokolech MTP, MT, MWB a MWBS. Zejména byla detailně analyzována komunikace, která je využívána při komunikaci mezi PC s aplikací Terminal a zařízením s OS RouterOS, nebo mezi dvěma zařízeními s OS MikroTik RouterOS. Tato komunikace využívá protokolů MTP a MT. Vlastnosti této komunikace splňují všechny požadavky kladené na automatickou vzdálenou správu zařízení s OS MikroTik RouterOS. Použití sestavy těchto protokolů bylo tedy vyhodnoceno jako nejlepší varianta řešení.

Vzhledem ke značnému rozsahu protokolu MWB a na něm založeného protokolu MWBS, nebyla komunikace pomocí těchto dvou protokolů podrobněji zkoumána. Pro zajištění jejich popisu by bylo nutné analyzovat velké množství binárních dat (stovky kB), což ovšem z hlediska doby, která byla vyhrazena pro analýzu komunikace nebylo vůbec časově možné. Ve výsledku bude stejně nutné implementovat do koncového zařízení jen jeden konfigurační protokol. Vzhledem k omezeným HW zdrojům použité mobilní platformy, by nejspíše ani implementace protokolu MWB do daného embedded systému nebyla možná. Použití protokolu MWBS by přineslo ještě další navýšení HW zátěže v důsledku použitého šifrovacího protokolu. Aktivní komunikace protokoly MWB nebo MWBS by tedy zásadně ovlivnila chod celého embedded systému, do kterého je požadováno implementovat možnost vzdálené správy zařízení s OS MikroTik RouterOS. Proto bylo tedy od prvotního záměru realizovat automatickou vzdálenou správu zařízení s OS MikroTik RouterOS na základě protokolů MWB/MWBS upuštěno.

Jak již bylo uvedeno, pro praktickou realizaci automatické správy zařízení s OS MikroTik RouterOS byla vybrána sestava proprietárních protokolů MTP-MT. Pro

použití těchto protokolů bylo napsáno několik modulů v jazyce C, jež ve výsledku tvoří knihovnu umožňující provádět automatickou vzdálenou správu zařízení s OS MikroTik RouterOS. Pro snažší vývoj a ladění celé knihovny byla tato knihovna vyvíjena na platformě Win32, ale při psaní jednotlivých modulů byl brán ohled na pozdější co nejméně problémový přenos této knihovny na mobilní platformu.

Ve výsledku se podařilo realizovat rozhraní, které umožňuje po vhodné parametrizaci jednoho stavového automatu v jednom z modulů knihovny, který stojí hierarchicky nejvýše, jakoukoliv požadovanou podporovanou operaci v OS MikroTik RouterOS. Což je velice výhodné, protože navržené a realizované řešení tak není omezeno jen na několik vybraných operací, jak se zprvu očekávalo u řešení založeného na použití protokolu MWB. Realizované řešení eliminovalo i nevhodnou vlastnost použití sestavy protokolů MTP-MT, která spočívá v tom, že v jeden okamžik je možné pomocí jednoho spojení vykonávat pouze jednu operaci. Uvedená negativní vlastnost je eliminována tím, že zvolená implementace uvedených protokolů umožňuje otevřít a spravovat několik spojení (session) současně. Schopnost spravovat více spojení současně byla úspěšně ověřena na 100 současně otevřených spojení. Funkčnost navrženého univerzálního rozhraní byla ověřena na úloze měření datových toků na vybraném síťovém rozhraní cílového zařízení. Všeobecná funkčnost sestavy protokolů MTP-MT byla ověřena na mnohahodinovém intenzivním provozu připojení ke vzdálenému systému pomocí této sestavy protokolů.

Seznam použitých zdrojů

[1] MikroTik. *RouterBoard RB411AH* [online]. c2011 [cit. 2011-05-05]. Dostupné z WWW: <http://routerboard.com/img/pricelist/83_m.png>.

[2] DOSTÁLEK, L.; KABELOVÁ, A. *Velký průvodce protokoly TCP/IP a systémem DNS*. 2. aktualizované vydání. Praha: Computer Press, 2000. 426 s. ISBN 80-7226-323-4.

[3] MikroTik. *MikroTik Wiki* [online]. 2010 [cit. 2010-12-05]. Dostupné z WWW: <http://wiki.mikrotik.com/wiki/Main_Page>.

[4] BRADEN, R. *RFC 1122 – Requirements for Internet Hosts – Communication Layers* [online]. 1989 [cit. 2010-12-05]. Dostupné z WWW: <http://tools.ietf.org/html/rfc1122>.

[5] *RFC 791 – Internet Protocol* [online]. 1981 [cit. 2010-12-05]. Dostupné z WWW: <http://tools.ietf.org/html/rfc791>.

[6] NICHOLS, K., et al. *RFC 2474 – Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* [online]. 1998 [cit. 2010-12-05]. Dostupné z WWW: <http://tools.ietf.org/html/rfc2474>.

[7] RAMAKRISHNAN, K.; FLOYD, S.; BLACK, D. *RFC 3168 – The Addition of Explicit Congestion Notification (ECN) to IP* [online]. 2001 [cit. 2010-12-05]. Dostupné z WWW: <http://tools.ietf.org/html/rfc3168>.

[8] BRADEN, R.; BORMAN, D.; PARTRIDGE, C. *RFC 1071 – Computing the Internet Checksum* [online]. 1988 [cit. 2010-12-05]. Dostupné z WWW: <http://tools.ietf.org/html/rfc1071>.

[9] POSTEL, J. *RFC 768 – User Datagram Protocol* [online]. 1980 [cit. 2010-12-05]. Dostupné z WWW: <http://tools.ietf.org/html/rfc768>.

[10] DEUTSCH, L. P. *RFC1321-based (RSA-free) MD5 library* [online]. Aladdin Enterprises, 2002-04-13 [cit. 2010-12-05]. Dostupné z WWW: <http://sourceforge.net/projects/libmd5-rfc/files/libmd5-rfc/2002-04-13/md5.tar.gz/download>.

Seznam použitých zkratek a symbolů

Zkratka	Význam
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SSH	Secure Shell
SNMP	Simple Network Management Protocol
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
TLS	Transport Layer Security
CDP	Cisco Discovery Protocol
MNDP	MikroTik Neighbor Discovery Protocol
MTP	MikroTik Transmission Protocol
MT	MikroTik Terminal
MWB	MikroTik WinBox
MWBS	MikroTik WinBox Secure
MTU	Maximum Transmission Unit
API	Application Programming Interface
OS	Operační systém

Seznam příloh

Příloha A

Příklad komunikace využívající sestavu protokolů MTP-MT

Propojení zařízení	přímé pomocí IEEE 802.3 Ethernet 100Base-TX
Klient	PC s aplikací Terminal
Server	MikroTik RouterBoard RB411AH s OS MikroTik RouterOS 4.11
MAC adresa klienta	00:10:5a:ca:81:af
MAC adresa serveru	00:0c:42:4c:d7:6e
Uživatelské jméno	admin+cte
Heslo	
Provedené operace	přihlášení ke vzdálenému systému, zápis a potvrzení příkazu `quit`

```
PC->RB: OPEN, 0 bytes send, session number: 0x0510, session protocol: 0x0015 (MikroTik Terminal)
Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.000000 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 00 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 00                                 ......

RB->PC: ACK 0 bytes (OPEN in frame 1), session number: 0x0510, session protocol: 0x0015
Frame 2: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.029539 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 00                                 ......

PC->RB: DATA subsequent to 0 bytes, 9 bytes send
        ctrlsq: START AUTHORIZATION (0 bytes)
Frame 3: 73 bytes on wire (584 bits), 73 bytes captured (584 bits), Time: 0.030160 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (31 bytes)

0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 00 56 34 12 ff 00 00 00 00 00      ......V4.......

RB->PC: ACK 9 bytes (DATA in frame 3)
Frame 4: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.030425 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 09                                 ......

RB->PC: DATA subsequent to 0 bytes, 25 bytes send
        ctrlsq: SALT STRING (16 bytes): 0x07b3471beb3917c43f80870b01b7c4f3
Frame 5: 89 bytes on wire (712 bits), 89 bytes captured (712 bits), Time: 0.048129 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (47 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 00 56 34 12 ff 01 00 00 00 10 07   ......V4........
0020  b3 47 1b eb 39 17 c4 3f 80 87 0b 01 b7 c4 f3      .G..9..?.......
```

PC->RB: ACK 25 bytes (DATA in frame 5)
Frame 6: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.048251 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 19                                 ......
```

PC->RB: DATA subsequent to 9 bytes, 80 bytes send
ctrlsq: AUTHORIZATION STRING (17 bytes): 0x00903c217f606a56671ac9f41539170bf3
ctrlsq: USERNAME (9 bytes): 0x61646d696e2b637465 (admin+cte)
ctrlsq: TERMINAL MODE (5 bytes): 0x6c696e7578 (linux)
ctrlsq: TERMINAL WIDTH (2 bytes): 0x5000 (80)
ctrlsq: TERMINAL HEIGHT (2 bytes): 0x1800 (24)
Frame 7: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits), Time: 0.048334 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (102 bytes)

```
0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 09 56 34 12 ff 02 00 00 00 11 00   ......V4........
0020  90 3c 21 7f 60 6a 56 67 1a c9 f4 15 39 17 0b f3   .<!.`jVg....9...
0030  56 34 12 ff 03 00 00 00 09 61 64 6d 69 6e 2b 63   V4.......admin+c
0040  74 65 56 34 12 ff 04 00 00 00 05 6c 69 6e 75 78   teV4.......linux
0050  56 34 12 ff 05 00 00 00 02 50 00 56 34 12 ff 06   V4.......P.V4...
0060  00 00 00 02 18 00                                 ......
```

RB->PC: ACK 89 bytes (DATA in frame 7)
Frame 8: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.048557 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 59                                 .....Y
```

RB->PC: DATA subsequent to 25 bytes, 9 bytes send
ctrlsq: AUTHORIZATION COMPLETED (0 bytes)
Frame 9: 73 bytes on wire (584 bits), 73 bytes captured (584 bits), Time: 0.057687 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (31 bytes)

```
0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 19 56 34 12 ff 09 00 00 00 00      ......V4.......
```

PC->RB: ACK 34 bytes (DATA in frame 9)
Frame 10: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.057767 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 22                                 .....".
```

RB->PC: DATA subsequent to 34 bytes, 555 bytes send
text to console:

```
CR CR LF CR CR LF CR CR LF CR CR LF CR CR LF CR CR LF CR LF
CR  MMM       MMM       KKK                          TTTTTTTTTTT      KKKCR LF
CR  MMMM     MMMM       KKK                          TTTTTTTTTTT      KKKCR LF
CR  MMM MMMM MMM  III  KKK  KKK  RRRRRR     OOOOOO      TTT     III  KKK  KKKCR LF
CR  MMM  MM  MMM  III  KKKKK     RRR  RRR  OOO  OOO     TTT     III  KKKKKCR LF
CR  MMM      MMM  III  KKK KKK   RRRRRR    OOO  OOO     TTT     III  KKK KKKCR LF
CR  MMM      MMM  III  KKK  KKK  RRR  RRR   OOOOOO      TTT     III  KKK  KKKCR LF
CR CR LF
CR  MikroTik RouterOS 4.11 (c) 1999-2010       http://www.mikrotik.com/CR LF
CR CR LF
```

Frame 11: 619 bytes on wire (4952 bits), 619 bytes captured (4952 bits), Time: 0.096381 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)

```
Data (577 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 22 0d 0d 0a 0d 0d 0a 0d 0d 0a 0d   .....".........
0020  0d 0a 0d 0d 0a 0d 0d 0a 0d 0d 0a 0d 0a 0d 20 20   ..............
0030  4d 4d 4d 20 20 20 20 20 20 4d 4d 4d 20 20 20 20   MMM      MMM
0040  20 20 20 4b 4b 4b 20 20 20 20 20 20 20 20 20 20      KKK
0050  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0060  54 54 54 54 54 54 54 54 54 54 54 20 20 20 20 20   TTTTTTTTTTT
0070  20 4b 4b 4b 0d 0a 0d 20 20 4d 4d 4d 4d 20 20 20    KKK...  MMMM
0080  20 4d 4d 4d 4d 20 20 20 20 20 20 20 4b 4b 4b 20    MMMM       KKK
0090  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00a0  20 20 20 20 20 20 20 20 20 54 54 54 54 54 54 54             TTTTTTT
00b0  54 54 54 54 20 20 20 20 20 20 4b 4b 4b 0d 0a 0d   TTTT      KKK...
00c0  20 20 4d 4d 4d 20 4d 4d 4d 4d 20 4d 4d 4d 20 20     MMM MMMM MMM
00d0  49 49 49 20 20 4b 4b 4b 20 20 4b 4b 4b 20 20 52   III  KKK  KKK  R
00e0  52 52 52 52 52 20 20 20 20 20 4f 4f 4f 4f 4f 4f   RRRRR     OOOOOO
00f0  20 20 20 20 20 20 54 54 54 20 20 20 20 20 49 49        TTT     II
0100  49 20 20 4b 4b 4b 20 20 4b 4b 4b 0d 0a 0d 20 20   I  KKK  KKK...
0110  4d 4d 4d 20 20 4d 4d 20 20 4d 4d 4d 20 20 49 49   MMM  MM  MMM  II
0120  49 20 20 4b 4b 4b 4b 20 20 20 20 20 52 52 52    I  KKKKK     RRR
0130  20 20 52 52 52 20 20 4f 4f 4f 20 20 4f 4f 4f 20     RRR  OOO  OOO
0140  20 20 20 20 54 54 54 20 20 20 20 20 49 49 49 20        TTT     III
0150  20 4b 4b 4b 4b 4b 0d 0a 0d 20 20 4d 4d 4d 20 20    KKKKK...  MMM
0160  20 20 20 20 4d 4d 4d 20 20 49 49 49 20 20 4b 4b       MMM  III  KK
0170  4b 20 4b 4b 4b 20 20 20 52 52 52 52 52 52 20 20   K KKK   RRRRRR
0180  20 20 4f 4f 4f 20 20 4f 4f 4f 20 20 20 20 20 54     OOO  OOO     T
0190  54 54 20 20 20 20 20 49 49 49 20 20 4b 4b 4b 20   TT     III  KKK
01a0  4b 4b 4b 0d 0a 0d 20 20 4d 4d 4d 20 20 20 20 20   KKK...  MMM
01b0  20 4d 4d 4d 20 20 49 49 49 20 20 4b 4b 4b 20 20    MMM  III  KKK
01c0  4b 4b 4b 20 20 52 52 52 20 20 52 52 52 20 20 20   KKK  RRR  RRR
01d0  4f 4f 4f 4f 4f 4f 20 20 20 20 20 20 54 54 54 20   OOOOOO      TTT
01e0  20 20 20 20 49 49 49 20 20 4b 4b 4b 20 20 4b 4b      III  KKK  KK
01f0  4b 0d 0a 0d 0d 0a 0d 20 20 4d 69 6b 72 6f 54 69   K......  MikroTi
0200  6b 20 52 6f 75 74 65 72 4f 53 20 34 2e 31 31 20   k RouterOS 4.11
0210  28 63 29 20 31 39 39 39 2d 32 30 31 30 20 20 20   (c) 1999-2010
0220  20 20 20 20 68 74 74 70 3a 2f 2f 77 77 77 2e 6d       http://www.m
0230  69 6b 72 6f 74 69 6b 2e 63 6f 6d 2f 0d 0a 0d 0d   ikrotik.com/....
0240  0a                                                .

PC->RB: ACK 589 bytes (DATA in frame 11)
Frame 12: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.096567 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 4d                                 .....M

RB->PC: DATA subsequent to 589 bytes, 105 bytes send
        text to console:
CR CR CR CR CR
CR[admin@MikroTik] >                                                CR
CR[admin@MikroTik] >
Frame 13: 169 bytes on wire (1352 bits), 169 bytes captured (1352 bits), Time: 0.115434 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (127 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 4d 0d 0d 0d 0d 0d 0d 5b 61 64 6d   .....M......[adm
0020  69 6e 40 4d 69 6b 72 6f 54 69 6b 5d 20 3e 20 20   in@MikroTik] >
0030  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0040  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0050  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0060  20 20 20 20 20 20 20 20 20 20 20 0d 5b 61 64 6d              .[adm
0070  69 6e 40 4d 69 6b 72 6f 54 69 6b 5d 20 3e 20      in@MikroTik] >

PC->RB: ACK 694 bytes (DATA in frame 13)
Frame 14: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.115589 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)
```

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 b6                                 ......
```

RB->PC, DATA subsequent to 694 bytes, 3 bytes send
text to console: ESC[r (enable scrolling for entire display)

```
Frame 15: 67 bytes on wire (536 bits), 67 bytes captured (536 bits), Time: 0.869785 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (25 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 b6 1b 5b 72                       .......[r
```

PC->RB: ACK 697 bytes (DATA in frame 15)

```
Frame 16: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 0.869991 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 b9                                 ......
```

RB->PC: ACK 89 bytes (keep-alive)

```
Frame 17: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 10.009510 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 59                                 .....Y
```

PC->RB: ACK 697 bytes (keep-alive)

```
Frame 18: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 10.016494 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 b9                                 ......
```

PC->RB: ACK 697 bytes (keep-alive)

```
Frame 19: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 20.016732 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 b9                                 ......
```

RB->PC: ACK 89 bytes (keep-alive)

```
Frame 20: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 20.017026 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 59                                 .....Y
```

PC->RB: DATA subsequent to 89 bytes, 22 bytes send
ctrlsq: TERMINAL WIDTH (2 bytes): 0x5000 (80)
ctrlsq: TERMINAL HEIGHT (2 bytes): 0x1800 (24)

```
Frame 21: 86 bytes on wire (688 bits), 86 bytes captured (688 bits), Time: 26.200844 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (44 bytes)

0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 59 56 34 12 ff 05 00 00 00 02 50   .....YV4.......P
0020  00 56 34 12 ff 06 00 00 00 02 18 00               .V4.........
```

RB->PC: ACK 111 bytes (DATA in frame 21)
Frame 22: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 26.201165 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 6f                                .....o
```

PC->RB: ACK 697 bytes (keep-alive)
Frame 23: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 30.016916 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 b9                                ......
```

RB->PC: ACK 111 bytes (keep-alive),
Frame 24: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 30.017191 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 6f                                .....o
```

PC->RB: DATA subsequent to 111 bytes, 1 byte send
text from console: g
Frame 25: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 33.723808 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

```
0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 6f 71                             .....oq
```

RB->PC: ACK 112 bytes (DATA in frame 25)
Frame 26: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 33.724111 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 70                                .....p
```

RB->PC: DATA subsequent to 697 bytes, 1 byte send
text to console: g
Frame 27: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 33.724818 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

```
0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 b9 71                             ......q
```

PC->RB: ACK 698 bytes (DATA in frame 27)
Frame 28: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 33.724991 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 ba                                ......
```

PC->RB: DATA subsequent to 112 bytes, 1 byte send
text from console: u
Frame 29: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 34.169116 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)

```
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 70 75                              .....pu
```

RB->PC: ACK 113 bytes (DATA in frame 29)

```
Frame 30: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 34.169381 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 71                                 .....q
```

RB->PC: DATA subsequent to 698 bytes, 1 byte send
text to console: u

```
Frame 31: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 34.170061 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 ba 75                              ......u
```

PC->RB: ACK 699 bytes (DATA in frame 31)

```
Frame 32: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 34.170233 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 bb                                 ......
```

PC->RB: DATA subsequent to 113 bytes, 1 byte send
text from console: i

```
Frame 33: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 34.922557 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 71 69                              .....qi
```

RB->PC: ACK 114 bytes (DATA in frame 33)

```
Frame 34: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 34.922858 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 72                                 .....r
```

RB->PC: DATA subsequent to 699 bytes, 1 byte send
text to console: i

```
Frame 35: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 34.923517 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 bb 69                              ......i
```

PC->RB: ACK 700 bytes (DATA in frame 35)

```
Frame 36: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 34.923708 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)
```

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 bc                                ......
```

PC->RB: DATA subsequent to 114 bytes, 1 byte send
text from console: t

```
Frame 37: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 35.499615 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 72 74                              .....rt
```

RB->PC: ACK 115 bytes (DATA in frame 37)

```
Frame 38: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 35.499896 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 73                                 .....s
```

RB->PC: DATA subsequent to 700 bytes, 1 byte send
text to console: t

```
Frame 39: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 35.500565 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 bc 74                              ......t
```

PC->RB: ACK 701 bytes (DATA in frame 39)

```
Frame 40: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 35.500736 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 bd                                 ......
```

PC->RB: DATA subsequent to 115 bytes, 1 byte send
text from console: [CR]

```
Frame 41: 65 bytes on wire (520 bits), 65 bytes captured (520 bits), Time: 37.838088 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (23 bytes)

0000  01 01 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 73 0d                              .....s.
```

RB->PC: ACK 116 bytes (DATA in frame 41)

```
Frame 42: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 37.838396 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

0000  01 02 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 74                                 .....t
```

RB->PC: DATA subsequent to 701 bytes, 26 bytes send
text to console: [CR][admin@MikroTik] > quit[CR][LF]

```
Frame 43: 90 bytes on wire (720 bits), 90 bytes captured (720 bits), Time: 37.839409 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (48 bytes)

0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 bd 0d 5b 61 64 6d 69 6e 40 4d 69   .......[admin@Mi
0020  6b 72 6f 54 69 6b 5d 20 3e 20 71 75 69 74 0d 0a   kroTik] > quit..
```

PC->RB: ACK 727 bytes (DATA in frame 43)
Frame 44: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 37.839582 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 d7                                 ......
```

RB->PC: DATA subsequent to 727 bytes, 13 bytes send
text to console: CR interrupted LF
Frame 45: 77 bytes on wire (616 bits), 77 bytes captured (616 bits), Time: 37.840794 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (35 bytes)

```
0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 d7 0d 69 6e 74 65 72 72 75 70 74   .......interrupt
0020  65 64 0a                                         ed.
```

PC->RB: ACK 740 bytes (DATA in frame 45)
Frame 46: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 37.840936 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 e4                                 ......
```

RB->PC: DATA subsequent to 740 bytes, 10 bytes send
text to console: ESC 7 (save cursor position and attributes)
ESC [;24r (set top and bottom margins)
ESC 8 (restore cursor position and attributes)
Frame 47: 74 bytes on wire (592 bits), 74 bytes captured (592 bits), Time: 37.841462 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (32 bytes)

```
0000  01 01 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 02 e4 1b 37 1b 5b 3b 32 34 72 1b 38   .......7.[;24r.8
```

PC->RB: ACK 750 bytes (DATA in frame 47)
Frame 48: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 37.841591 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 02 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 02 ee                                 ......
```

RB->PC: CLOSE, session number: 0x0510, session protocol: 0x0015 (MikroTik Terminal)
Frame 49: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 37.884624 s
Ethernet II, Src: Routerbo_4c:d7:6e (00:0c:42:4c:d7:6e), Dst: 3com_ca:81:af (00:10:5a:ca:81:af)
Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 ff 00 0c 42 4c d7 6e 00 10 5a ca 81 af 00 15   ....BL.n..Z.....
0010  05 10 00 00 00 00                                 ......
```

PC->RB: CLOSE, session number: 0x0510, session protocol: 0x0015 (MikroTik Terminal)
Frame 50: 64 bytes on wire (512 bits), 64 bytes captured (512 bits), Time: 37.884796 s
Ethernet II, Src: 3com_ca:81:af (00:10:5a:ca:81:af), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol, Src: 192.168.88.10 (192.168.88.10), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: 20561 (20561), Dst Port: 20561 (20561)
Data (22 bytes)

```
0000  01 ff 00 10 5a ca 81 af 00 0c 42 4c d7 6e 05 10   ....Z.....BL.n..
0010  00 15 00 00 00 00                                 ......
```

Příloha B

Časové intervaly, po kterých došlo k přeposlání MTP segmentu typu OPEN při jeho nepotvrzení MTP segmentem typu ACK

			Interval [ms]			
Pokus	**Čas [s]**	**sessnm**	**č. 1**	**č. 2**	**č. 3**	**č. 4**
1	0,0	0x09E4	112	211	400	807
2	4,8	0x09B0	102	199	411	808
3	9,6	0x0EB8	99	216	396	813
4	14,3	0x009C	105	213	397	814
5	19,0	0x0868	101	214	400	810
6	23,8	0x0168	110	200	406	812
7	28,5	0x0650	114	203	408	807
8	33,4	0x0FA0	105	214	396	814
9	38,2	0x0F60	108	216	394	814
10	43,0	0x0304	110	212	396	812

Tabulka 1: Klient Terminal

			Interval [ms]	
Pokus	**Čas [s]**	**sessnm**	**č. 1**	**č. 2**
1	0,0	0xEA9B	113	106
2	2,5	0xED9B	112	111
3	5,4	0xEF9B	108	112
4	7,8	0xF29B	96	109
5	10,0	0xF49B	114	106
6	12,4	0xF69B	110	112
7	15,0	0xF99B	99	109
8	17,2	0xFB9B	104	109
9	19,4	0xFD9B	111	107
10	21,5	0x009C	111	107

Tabulka 2: Klient WinBox 2.2.16 (MikroTik RouterOS 4.11)

			Interval [ms]					
Pokus	**Čas [s]**	**sessnm**	**č. 1**	**č. 2**	**č. 3**	**č. 4**	**č. 5**	**č. 6**
1	0,0	0x0364	20	30	50	90	170	330
2	8,3	0x0365	20	30	50	90	170	330
3	15,2	0x0366	20	30	50	90	170	330
4	32,3	0x0367	20	30	50	90	170	330
5	39,5	0x0368	20	30	50	90	170	330
6	45,4	0x0369	20	30	50	90	170	330
7	51,4	0x036a	20	30	50	90	170	330
8	60,9	0x036b	20	30	50	90	170	330
9	67,2	0x036c	20	30	50	90	170	330
10	78,5	0x036d	20	30	50	90	170	330

Tabulka 3: Klient pro MT v MikroTik RouterOS 4.11

(připojení ke klientovi přes sériové rozhraní RS-232, použití příkazu konzole)

			Interval [ms]					
Pokus	**Čas [s]**	**sessnm**	**č. 1**	**č. 2**	**č. 3**	**č. 4**	**č. 5**	**č. 6**
1	0,0	0x035F	20	30	50	90	170	330
2	73,7	0x0360	20	30	50	90	170	330
3	130,0	0x0361	20	30	50	90	170	330
4	142,9	0x0362	20	30	50	90	170	330
5	154,0	0x0363	20	30	50	90	170	330
6	176,0	0x0364	20	30	50	90	170	330
7	186,2	0x0365	20	30	50	90	170	330
8	200,7	0x0366	20	30	50	90	170	330
9	209,8	0x0367	20	30	50	90	170	330
10	221,8	0x0368	20	30	50	90	170	330

Tabulka 4: Klient pro MT v MikroTik RouterOS 4.11

(připojení ke klientovi přes WinBox 2.2.16, použití funkce z menu)

			Interval [ms]					
Pokus	**Čas [s]**	**sessnm**	**č. 1**	**č. 2**	**č. 3**	**č. 4**	**č. 5**	**č. 6**
1	0,0	0x0363	18	30	50	90	170	330
2	105,6	0x0364	16	30	50	90	170	330
3	111,7	0x0365	11	30	50	90	170	330
4	117,8	0x0366	11	30	50	90	170	330
5	125,3	0x0367	18	30	50	90	170	330
6	130,4	0x0368	18	30	50	90	170	330
7	135,1	0x0369	18	30	50	90	170	330
8	141,6	0x036a	11	30	50	90	170	330
9	147,4	0x036b	16	30	50	90	170	330
10	152,7	0x036c	15	30	50	90	170	330

Tabulka 5: Klient pro MT v MikroTik RouterOS 4.11

(připojení ke klientovi přes WinBox 2.2.16, použití příkazu konzole)

Příloha C

Mapování kláves do protokolu MT

Klávesa	HEX
???	00
CTRL + a	01
CTRL + b	02
CTRL + c	03
CTRL + P/B	
CTRL + d	04
CTRL + e	05
CTRL + f	06
CTRL + g	07
CTRL + h	08
BACKSPACE	
CTRL + i	09
TAB	
CTRL + j	0A
CTRL + ENT	
CTRL + k	0B
CTRL + l	0C
CTRL + m	0D
ENTER	
CTRL + n	0E
CTRL + o	0F
CTRL + p	10
CTRL + q	11
CTRL + r	12
CTRL + s	13
CTRL + t	14
CTRL + u	15
CTRL + v	16
CTRL + w	17
CTRL + x	18
CTRL + y	19
CTRL + z	1A
CTRL + [	1B
ESC	

Klávesa	HEX
CTRL + \	1C
???	1D
???	1E
???	1F
SPACE	20
!	21
"	22
#	23
$	24
%	25
&	26
'	27
(	28
)	29
*	2A
+	2B
,	2C
-	2D
.	2E
/	2F
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39
:	3A
;	3B
<	3C
=	3D

Klávesa	HEX
>	3E
?	3F
@	40
A	41
B	42
C	43
D	44
E	45
F	46
G	47
H	48
I	49
J	4A
K	4B
L	4C
M	4D
N	4E
O	4F
P	50
Q	51
R	52
S	53
T	54
U	55
V	56
W	57
X	58
Y	59
Z	5A
[	5B
/	5C
]	5D
^	5E
	5F

Klávesa	HEX
`	60
a	61
b	62
c	63
d	64
e	65
f	66
g	67
h	68
i	69
j	6A
k	6B
l	6C
m	6D
n	6E
o	6F
p	70
q	71
r	72
s	73
t	74
u	75
v	76
w	77
x	78
y	79
z	7A
{	7B
|	7C
}	7D
~	7E
DELETE	7F
CTRL + BS	

Klávesa	HEX		
F1	1B	4F	50
F2	1B	4F	51
F3	1B	4F	52
F4	1B	4F	53
F5	1B	4F	74
F6	1B	4F	75
F7	1B	4F	76
F8	1B	4F	6C
F9	1B	4F	77
F10	1B	4F	78

Klávesa	HEX			
↑	1B	5B	41	
↓	1B	5B	42	
→	1B	5B	43	
←	1B	5B	44	
HOME	1B	5B	31	7E
END	1B	5B	34	7E
PageUp	1B	5B	35	7E
PageDown	1B	5B	36	7E

`CTRL + ]` vykoná uzavření spojení klientem (znak `0x1D` se ale serveru neodesílá)
`CTRL + d` vykoná uzavření spojení serverem (znak `0x04` se serveru odesílá)
`CTRL + c` nebo `CTRL + Pause/Break` vykoná ukončení prováděného příkazu