

UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Multipatformní nástroj pro sdílení souborů na síti

Jan Šmíd

Diplomová práce

2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jan Šmíd**
Osobní číslo: **I14288**
Studijní program: **N2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **Multiplatformní nástroj pro sdílení souborů na síti**
Zadávající katedra: **Katedra softwarových technologií**

Z á s a d y p r o v y p r a c o v á n í :

Teoretická část popíše proces návrhu síťového protokolu a existující protokoly pro sdílení (např. NFS, CIFS). Vysvětlí vlastnosti uvedených protokolů a rozdíly mezi nimi. Vznikne návrh nového protokolu umožňujícího aplikaci sdílení souborů na lokální síti s minimálními nároky na znalosti uživatelů. Návrh se porovná se zkoumanými protokoly.

Praktická část implementuje multiplatformní nástroj (minimálně pro Linux a Windows), který bude daný protokol používat, a to včetně GUI. Nástroj umožní uživateli sdílet vybrané soubory a nastavit omezení pro ostatní uzly v síti.

Rozsah grafických prací:

Rozsah pracovní zprávy: cca 40–50 stran

Forma zpracování diplomové práce: tištěná

Seznam odborné literatury:

**NFS Version 3 Protocol Specification* [online]. 1995 [cit. 2015-10-01].

URL: <https://tools.ietf.org/pdf/rfc1813>

**Network File System (NFS) version 4 Protocol* [online]. 1995 [cit. 2015-10-01]. URL:

<https://tools.ietf.org/pdf/rfc3530>

*[*MS-CIFS*]: *Common Internet File System (CIFS) Protocol* [online]. Microsoft, October 16 2015 [cit. 2015-10-16]. Dostupné jako HTML či PDF.

URL: <https://msdn.microsoft.com/en-us/library/ee442092.aspx>

**Qt Project* [online]. 2014, [cit. 2014-10-06]. Dokumentace projektu QT. URL:

<http://qt-project.org/doc/>

*STONES, Richard – MATTHEW, Neil: *Linux: Začínáme programovat*. Praha: Computer Press, 2000. ISBN 80-7226-307-2.

*STONES, Richard – MATTHEW, Neil: *Linux: Programujeme profesionálně*. Praha: Computer Press, 2001. ISBN 80-7226-532-6.

**Linux: Dokumentační projekt*. 4. aktualizované vydání. Brno: Computer Press, 2008. 1336 s. ISBN: 978-80-251-1525-1. Dostupné též online:

<http://www.root.cz/knihy/linux-dokumentacni-projekt-4-vydani/>

Vedoucí diplomové práce: Mgr. Tomáš Hudec

Katedra informačních technologií

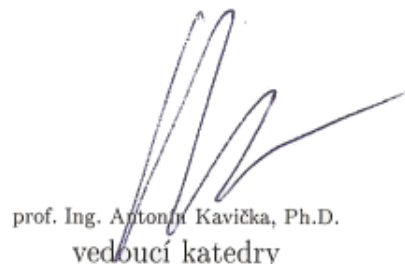
Datum zadání diplomové práce: 31. října 2016

Termín odevzdání diplomové práce: 17. května 2017



Ing. Zdeněk Němec, Ph.D.
děkan

L.S.



prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2016

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury. Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 14. 8. 2017

Poděkování

Rád bych poděkoval Mgr. Tomášovi Hudcovi za odborné vedení této práce a zejména za opětovné probuzení mého zájmu o svobodné technologie a operační systémy, se kterými seznamoval své studenty během let našeho studia.

Anotace

Teoretická část popíše proces návrhu síťového protokolu a existující protokoly pro sdílení (např. NFS, CIFS). Vysvětlí vlastnosti uvedených protokolů a rozdíly mezi nimi. Vznikne návrh nového protokolu umožňujícího aplikaci sdílení souborů na lokální síti s minimálními nároky na znalosti uživatelů. Návrh se porovná se zkoumanými protokoly. Praktická část implementuje multiplatformní nástroj (minimálně pro Linux a Windows), který bude daný protokol používat, a to včetně GUI. Nástroj umožní uživateli sdílet vybrané soubory a nastavit omezení pro ostatní uzly v síti.

Klíčová slova

soubor, sdílení, síť, protokol, návrh, implementace

Title

Multiplatform Tool for Network File Sharing

Annotation

The theoretical part describes the network protocol design process and existing sharing protocols (e.g. NFS, CIFS). A new protocol will be created to allow file sharing on the local network with minimal user knowledge requirements. The proposal compares with the examined protocols. The practical part implements a multiplatform tool (at least for Linux and Windows) based on newly created protocol, graphical user interface included. The tool will allow the user to share the selected files and set restrictions for other nodes on the network.

Keywords

file, sharing, network, protocol, draft, implementation

Obsah

1	Server Message Block	14
1.0.1	SMB 2.x	14
1.0.2	SMB 3.x	15
1.1	Samba	15
1.2	Princip funkce SMB verze 2 a 3	16
1.2.1	Transport zpráv	16
1.2.2	Zprávy SMB 2	18
1.2.3	Navázání komunikace se SMB serverem	21
1.2.4	Čtení a zápis souborů pomocí SMB 2	24
1.2.5	Odpojení se od sdíleného zdroje	29
2	Network File System	30
2.1	Remote Procedure Call	32
2.1.1	Zprávy RPC	32
2.1.2	Mapování portů	34
2.2	NFS verze 4	38
2.2.1	Změny v protokolu NFSv3	38
2.2.2	Network File System 4.x	39
2.2.3	Nové vlastnosti NFSv4	40
2.2.4	Parallel NFS	44
3	Návrh vlastního protokolu	46
3.1	Easy Sharing Protocol	47
3.1.1	Služba poskytovaná protokolem	48
3.1.2	Předpoklady o prostředí	48
3.1.3	Slovník zpráv	49

3.1.4	Formát zpráv	49
3.1.5	Pravidla pro výměnu zpráv	50
4	Standardizace protokolu	60
4.1	Requests for Comments	60
4.2	Standards track	62
4.2.1	Standardizační akce	62
4.2.2	Proposed Standard	62
4.2.3	Draft Standard	63
4.2.4	Internet Standard	64
4.2.5	Historic	64
4.2.6	Experimental	64
4.2.7	Nestandardní úrovně vyspělosti	64
4.3	Standardizační proces	65
4.3.1	Vyvolání standardizační akce	65
4.3.2	Revize a schválení akce	65
4.3.3	Zveřejnění	66
4.3.4	Selhání procesu	66
4.4	Revidování standardu	66
4.5	Standardizační autority	67
4.5.1	Proprietární specifikace	67
4.5.2	Otevřené standardy	68
5	Demonstrační implementace	70
5.1	Vývojové knihovny	70
5.1.1	Vývoj a kompatibilita	71
5.1.2	Struktura aplikace	71

5.2	Ovládání aplikace	72
5.2.1	Spuštění aplikace	72
5.2.2	Sdílení souborů	73
5.2.3	Stahování souboru	74
6	Závěr	75

Seznam obrázků

1	RDMA over Converged Ethernet (Beck, 2014, s. 3)	18
2	Navázání SMB2 spojení	22
3	Zápis a čtení souboru	24
4	Odpojení sdíleného zdroje	29
5	Princip protokolu NFS (Sandberg, 1985, s. 4)	31
6	Portmap principle (Maiti, 2011, s. 22)	35
7	Postup kompilace lokálního a RPC programu	37
8	Theo de Rath k NFSv4 (Raadt, 2010)	40
9	Pseudo-file system	41
10	Parallel NFS (ZHANG, 2009, s. 7)	44
11	Peer-to-peer architektura	49
12	Zachycená zpráva HELLO	51
13	HELLO zprávy	52
14	Status v závislosti na časovačích	53
15	CRC a SUM zprávy	58
16	BYE zprávy	59
17	Schéma úrovní vyspělosti	63
18	Okno aplikace	70
19	Dokumentace funkce (Qt, 2017)	71
20	Start aplikace	73
21	Sdílení souborů	73
22	Kontrolní součty souboru	74

Seznam tabulek

1	Network model	17
2	SMB 2 Transport header	17
3	SMB 2 Message Header	20
4	Hodnoty dialektů pro NEGOTIATE Request/Response	22
5	SMB 2 SESSION_SETUP Response	23
6	Typy sdílených zdrojů	24
7	Hodnoty pole ShareAccess	25
8	Hodnoty pole CreateDisposition	25
9	Hodnoty pole CreateAction	26
10	Zpráva SMB2 SET_INFO Response	26
11	SMB 2 WRITE Request	28
12	SMB 2 WRITE Response	28
13	SMB 2 READ Response	29
14	TREE_DISCONNECT a LOGOFF zprávy	30
15	XDR reprezentace typu integer	32
16	RPC Call and Reply messages (Maiti, 2011, s. 22-29)	33
17	Rozdělení síťových portů	34
18	Seznam definovaných zpráv	50
19	Hodnoty zpráv a UTF-8	51
20	Příklady popisovače souboru	54
21	Vytvoření řetězce prvků	55
22	Kontrolní součty na systému Debian GNU/Linux	56
23	Kontrolní součty pomocí knihoven Qt	57

Seznam zkratek

SMB - Server Message Block

NFS - Network File System

LTS - Long Term Support

ESP - Easy Share Protocol

UDP - User Datagram Protocol

TCP - Transmission Control Protocol

VR - Virtuální Realita

Terminologie

implementace: Realizace teoretického návrhu. V programové úrovni zaměnitelné s aplikací.

protokol: Množina akcí a pravidel.

uzel: Pracovní stanice v lokální síti.

Úvod

Sdílení souborů je alfou a omegou počítačových sítí a internetu od nepaměti. Prakticky všechny síťové služby jsou na určité úrovni založené na sdílení souborů. Hitem posledních let je technologie skrývající se za reklamním sloganem jménem „cloud”. Ačkoli se vedle veřejných cloudů stále častěji objevují i ty privátní, stále se jedná o technologii sdílení a synchronizace zahrnující dedikovaný server, který slouží jako prostředník mezi připojenými uzly.

Pokud bychom ke sdílení přistupovali z pohledu obyčejného uživatele, jehož cílem je zkopírovat soubory z bodu A do bodu B, musíme notně dojít ke zjištění, že pro takové nasazení není cloud nejvhodnější volbou. Mnoho uživatelů může odradit už nutnost instalace klienta zajišťujícího synchronizaci, zatímco konfigurace vlastního cloud serveru je pro mnohé naprosto nepředstavitelná. Alternativou mohou být služby typu DropBox, nicméně zde přicházíme o výhody vysoké rychlosti lokálních sítí, jedná se přeci jen o veřejná úložiště dostupná přes internet. V případě sdílení větších souborů bychom k takovým úložištím museli přistupovat skrze vysokorychlostní připojení, které ale není široce dostupné. Poskytovatelé těchto úložišť by navíc s největší pravděpodobností negarantovali vysokorychlostní přístup všem neplatícím zákazníkům.

Nevhodnost těchto služeb je podtržena vysokým prodejem fyzických úložišť, jako jsou USB klíčenky a externí disky. Je až s podivem, že se v době běžně dostupných VR technologií a 3D tiskáren většina lidí stále uchyluje k vytváření nadbytečných kopií na externích médiích. Tato média jsou často velmi pomalá a v nezanedbatelném počtu případů jsou jen další zbytečnou investicí.

Systém Windows sice dlouhá léta disponuje vestavěnou funkcí sdílení souborů, nicméně v domácím prostředí není její použití obvyklé. Velké množství uživatelů preferuje právě velice rozšířené USB disky. Ačkoli není nastavení sdílení souborů nijak extra náročné, přesto uživatelé preferují prosté nakopírování souborů na přenosné úložiště. Jedním z důvodů také může být politika společnosti Microsoft, která aktualizacemi neohlášeně rozbíjí kompatibilitu starších a novějších edic Windows.

V prostředí mnoha linuxových distribucí je situace ještě o něco náročnější. Na rozdíl od Windows, kde je sdílení souborů doménou protokolu SMB/CIFS, v Linuxu je široce používán i protokol NFS. Ten je sice možné používat i ve Windows, nicméně je k tomu

nutné aktivovat tzv. „unixový subsystém”. To je koncept pro koncového uživatele naprosto neznámý, přičemž se takový uživatel bude po nejbližším USB disku rozhlížet již při zmínce o systému Unix.

Dalším problémem je uživatelské prostředí. To, co běžný uživatel rozličných linuxových distribucí vidí, je jen jedno z mnoha známých grafických prostředí. Každé z těchto prostředí si ale implementuje grafickou aplikaci pro sdílení souborů po svém. Samotné sdílení sice obstarává jedna a ta samá aplikace, co vidí uživatel si ale vývojáři každého prostředí implementují dle vlastního vědomí a svědomí. Mnoho především pokročilých uživatelů si raději konfiguruje jimi preferovanou serverovou aplikaci manuálně. To je výhodné především v momentech, kdy uživatel vyžaduje nastavení pokročilých vlastností, které grafické nadstavby běžně nenabízejí. Manuální konfigurace ale nemusí být nejvhodnější volbou, pokud je uživatel ve spěchu. Málokdo si například z paměti vybaví dostupné parametry direktiv konfiguračního souboru samba serveru. S valnou většinou linuxových aplikací se sice instalují i ukázkové soubory a manuálové stránky, ale ani ty nemusí vždy obsahovat požadované informace, popř. je nezobrazují ve vyhovující formě.

V této práci se proto budu zabývat popisem funkce výše zmíněných protokolů NFS a SMB/CIFS, na které bude navazovat návrh vlastního síťového protokolu pro sdílení souborů. V další části dojde k popisu implementace jednoduché multiplatformní aplikace využívající tento nový protokol.

1 Server Message Block

Protokol SMB vytvořil v roce 1983 doktor Barry Feigenbaum v korporaci IBM. Protokol byl zprvu pojmenovaný BAF, podle iniciál jeho autora, nicméně byl přejmenovaný ještě před prvním oficiálním vydáním. Cílem vzniku SMB bylo přeměnit DOSové přerušení 21h „Random read” pro přístup k lokálním souborům na síťový souborový systém (Tridgell, 2005).

Pojmenování SMB bývá volně zaměňováno za označení CIFS (Common Internet File System) z dílny společnosti Microsoft. CIFS ve skutečnosti značí jeden z „dialektů” protokolu SMB užívaných v produktech společnosti Microsoft.

SMB protokol byl jeho autorem navržen jako rozšiřitelný, čehož Microsoft v rámci jeho adaptování do svých produktů náležitě využil. V průběhu let (nejen) Microsoft do protokolu přidával nové vlastnosti, v důsledku čehož bylo potřeba tyto „edice” protokolu identifikovat a vzájemně odlišit. K rozlišení jednotlivých edic SMB Microsoft použil pojem „dialekt”. CIFS tedy není synonymem SMB, nejedná se ani o jeho „novou” verzi. CIFS je dialektem protokolu SMB, jeho rozšířenou edicí (Sharpe, 1996).

Identifikace dialektu neboli „dialect negotiation” je prvním krokem při komunikaci mezi SMB serverem a jeho klienty. Originální dialekt z dílny IBM a tedy doktora Feigenbauma je označován jako „PC NETWORK PROGRAM 1.0”. Od roku 1996 se Microsoft pokoušel vydat dialekt CIFS jako IETF standard pod označením „A Common Internet File System (CIFS/1.0) Protocol”. V rámci zvýšené otevřenosti Microsoft začal v té době pořádat každoroční CIFS konference, které měly pomoci popularizovat CIFS jako otevřený standard. IETF draft popisující CIFS/1.0 nicméně v polovině roku 1998 vyexpiroval a Microsoft upustil od snah protokol standardizovat a stáhl se i z vlastních konferencí (Sharpe, 1996).

1.0.1 SMB 2.x

V roce 2007 Microsoft vydal novou verzi SMB 2.0 jako součást tehdy „přelomových” Windows Vista. Ačkoli se jednalo o proprietární protokol, Microsoft vydal jeho specifikace pro veřejnost s cílem popularizovat tento protokol a umožnit spolupráci jiných systémů s produkty Microsoftu. Mezi největšími novinkami bylo snížení „upovídánosti” protokolu, kdy více než stovka příkazů podporovaná v SMB1 byla redukována na méně než 20 příkazů

v SMB2. Dále přibyla podpora pipeliningu, tedy řetězení příkazů a zvýšení výkonu u sítí s vysokou latencí, což souvisí právě s použitím pipeline (Barreto, 2008).

SMB2 totiž odesílal dodatečné příkazy přímo a bez čekání, zatímco SMB1 musel nejdříve čekat na odpověď protistrany. SMB1 tento problém řešil mechanismem AndX, nicméně ten se nerozšířil. Další novinkou byla schopnost překonat krátký výpadek připojení bez ukončení spojení, díky čemuž již nemuselo docházet k opakovanému navazování „duplicitních“ spojení. Mimo jiné přibyla podpora symbolických odkazů, podepisování zpráv pomocí hašovacího algoritmu SHA-256 a lepší škálovatelnost. V důsledku toho bylo možné zvýšit počet uživatelů, sdílených zdrojů a otevřených souborů v rámci jednoho serveru.

SMB2 také rozšířil adresní prostor a odstranil původní omezení pro velikost bloku, čímž se zvýšila výkonnost protokolu při přenášení velkých souborů po rychlých sítích. Protokol také zůstal zpětně kompatibilní s původním SMB. Servery implementující SMB2 tak mohly i nadále komunikovat se staršími klienty v rámci jednoho protokolu (Barreto, 2008).

1.0.2 SMB 3.x

V dalších vydáních SMB přibýly funkce zvyšující výkon a bezpečnost protokolu. Ve verzi 3.0 to byl například SMB MultiChannel, umožňující více spojení v rámci jedné relace, v 3.0.2 zase možnost deaktivovat podporu pro SMB1. Do verze 3.1.1 byly zahrnuty další algoritmy pro šifrování a autentizaci, např. AES 128 GCM (Microsoft, 2017, s. 22-27).

1.1 Samba

Samba je svobodnou implementací SMB serveru, o které se obecně tvrdí, že vznikla reverzním inženýrstvím existujících uzavřených řešení. Andrew Tridgell, zakladatel a jeden z hlavních vývojářů projektu Samba, se však s tímto tvrzením neztotožňuje. Zatímco s reverzním inženýrstvím je spojena analýza spustitelného kódu pomocí specializovaných nástrojů, Samba byla vytvořena „na zelené louce“ pomocí analýzy síťového provozu mezi existujícími implementacemi protokolu SMB (Tridgell, 1997).

Ačkoli je SMB dlouhodobě spojován se společností Microsoft, nebylo tomu tak vždy. Sambu tedy nelze považovat ani za klon implementace od Microsoftu. Znamená to pouze to, že klientské a serverové aplikace Microsoftu jsou při síťové analýze testovány častěji.

Samba prapůvodně vznikla na konci roku 1991 na Australské Národní Univerzitě v Canbeře pod prostým názvem „smbserver“. Když si v roce 1994 začala tento název nárokovat společnost Syntax stojící za produktem zvaným „TotalNet advanced Server“, byl název projektu z preventivních důvodů změněn. Toto přejmenování proběhlo poměrně kuriózním způsobem a to dotazem na `/usr/dict/words` na slova obsahující písmena S, M a B (Tridgell, 1998).

Čeho se tehdy Andrew Tridgell snažil dosáhnout, bylo připojení diskového prostoru ze systému Ultrix na jeho pracovní stanici od společnosti Sun. To bylo tehdy možné výhradně pomocí nástroje pathworks, který ale nebyl pro jeho stanici dostupný. Andrew tedy začal studovat pathworks a komunikaci jím užívaného protokolu a to metodou zachytávání zpráv. Po pár dnech byl Andrew schopen vytvořit jednoduchý program schopný nespolehlivého připojení k jeho serveru. Jeho program umožňoval základní souborové operace a umožnil mu tak pracovat s diskovým prostorem i z jeho Sunovské stanice.

Sám Andrew tehdy neměl tušení, že to, co vytvořil, bylo implementací protokolu SMB. Svě studium na univerzitě v Canbeře Andrew ukončil v roce 1999 dizertační prací, v níž se spolupodílel na vzniku algoritmu a nástroje rsync (Tridgell, 1999).

1.2 Princip funkce SMB verze 2 a 3

První kompletní specifikaci protokolu SMB 2 Microsoft vydal v lednu roku 2007. Za deset let vyšlo na padesát revizí tohoto dokumentu. Ačkoli číslo značící verzi protokolu vystoupalo na 3.1.1, autoři specifikace jej stále označují jako „SMB 2 Protocol“. Zatímco SMBv2 byl zcela nový a kompletně nezávislý na původním protokolu SMB/CIFS, další revize přinášely pouze dílčí úpravy zvyšující výkon a možnosti protokolu (Microsoft, 2017, s. 2-3). Tato povýšení Microsoft nadále označuje termíny dialekt a rodiny dialektů (Microsoft, 2017, s. 19-24).

1.2.1 Transport zpráv

Samotný SMB 2 funguje především na aplikační vrstvě, přičemž pro transport svých zpráv podporuje napříč různými dialekty přímé spojení přes TCP (Direct TCP), NetBIOS přes TCP a vzdálený přímý přístup do paměti RDMA (Remote Direct Memory Access) (Microsoft, 2017, s. 22).

Tabulka 1: Network model

<i>ISO/OSI model</i>				<i>TCP/IP model</i>
Application	SMB2			Application
Presentation				
Session		NetBIOS	RDMA	Transport
Transport	TCP/UDP	TCP/UDP		
Network	IP	IP		Network
Data Link	Ethernet	Ethernet	NIC	Network Access
Physical				

Direct TCP

Při přímé komunikaci SMB 2 užívá TCP (i UDP) port 445 (Microsoft, 2017, s. 27). Všechny zprávy proměnné délky pocházející z aplikační vrstvy předchází tzv. „Direct TCP header”. Jedná se o hlavičku s délkou 32 bitů, kde první bajt musí vždy nést hodnotu nula. Zbylé tři nesou informaci o délce transportované zprávy. Direct TCP je podporováno všemi dialekty SMB2 (Microsoft, 2017, s. 28).

Tabulka 2: SMB 2 Transport header

0...7	8...15	16...23	24...31
0x00	Length		
SMB 2 Message			
...			

NetBIOS over TCP

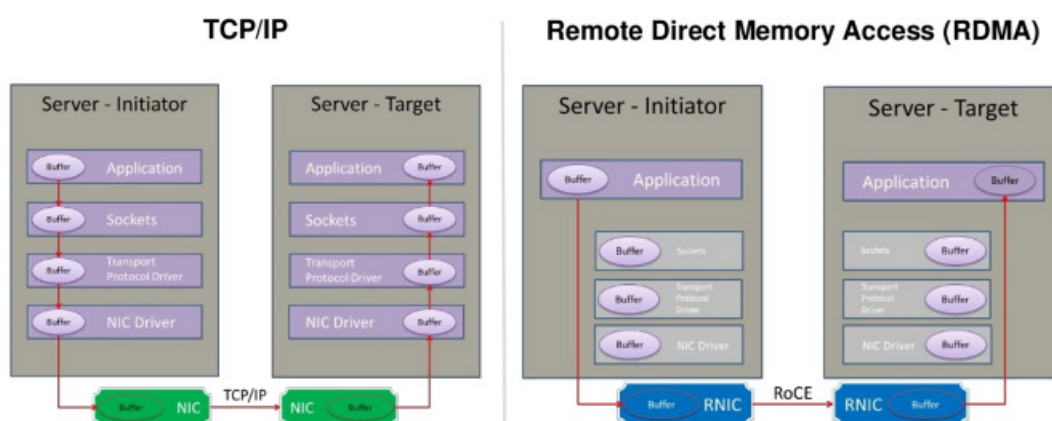
Komunikace přes NetBIOS je v protokolu zahrnuta především z důvodu kompatibility se staršími SMB servery a operačními systémy Windows implementujícími tyto servery. Podpora NetBIOSu je obsažena v dialektech 2.0.2, 2.1, 3.0, a 3.0.2. NetBIOS (Microsoft, 2017, s. 28) neboli „*Network Basic Input/Output System*” je API poskytující služby na páté (relační) vrstvě OSI modelu. Byl vytvořen v roce 1983 ve společnosti Sytek Inc. pro aplikace komunikující v lokální síti skrze technologie IBM. NetBIOS sloužil pro spojení i nespojovou komunikaci a překlad jmen přes sady protokolů DECnet, IPX a TCP/IP (Evans, 2003, s. 7). Nad TCP/IP NetBIOS používá porty

- 137 TCP/UDP - Name Service (NetBIOS-NS) - registrace a překlad jmen,
- 138 UDP - Datagram distribution service - nespojová komunikace,

- 139 TCP - Session service - spojitová komunikace (Evans, 2003, s. 38).

Remote Direct Memory Access

RDMA slouží k transportu SMB zpráv z paměti počítače přímo do paměti vzdáleného počítače bez zapojení služeb operačního systému. Takové transporty nekladou nároky na procesor, mezipaměť nebo přepínání kontextu a nejsou ovlivněny plánovačem procesů operačního systému. Vyžadují ale specializované síťové služby a protokoly, jako například RoCE (RDMA over Converged Ethernet) (Beck, 2014, s. 2).



Obrázek 1: RDMA over Converged Ethernet (Beck, 2014, s. 3)

1.2.2 Zprávy SMB 2

Komunikace SMB serveru s jeho klienty je založená na zasílání SMB zpráv, které jsou dle svého významu rozděleny do několika skupin. Definice a názvy těchto skupin i do nich zahrnutých zpráv se s každým novým dialektem mírně odlišují (Microsoft, 2017, s. 28-29). Jejich podrobný popis je nad rámec této práce, ale popis a princip využití vybraných zpráv bude uveden v následující podsekci.

- Protocol negotiation (SMB2 NEGOTIATE).
 - V rodině dialektu SMB 3.x rozšířena na *Protocol Negotiation and secure dialect validation* (SMB2 NEGOTIATE, SMB2 IOCTL).
- User authentication (SMB2 SESSION_SETUP, SMB2 LOGOFF).
- Share access (SMB2 TREE_CONNECT, SMB2 TREE_DISCONNECT).

- File access
 - (SMB2 CREATE, SMB2 CLOSE, SMB2 READ, SMB2 WRITE, SMB2 LOCK, SMB2 IOCTL, SMB2 QUERY_INFO, SMB2 SET_INFO, SMB2 FLUSH, SMB2 CANCEL).
- Directory access (SMB2 QUERY_DIRECTORY, SMB2 CHANGE_NOTIFY).
- Volume access (SMB2 QUERY_INFO, SMB2 SET_INFO).
- Cache coherency (SMB2 OPLOCK_BREAK).
- Simple messaging (SMB2 ECHO).
- Hash Retrieval (SMB2 IOCTL). Přidáno v rodině dialektu 2.x a rozšířeno v 3.x.
- Encryption (SMB2 TRANSFORM_HEADER). Přidáno v rodině dialektu 3.x.

Hlavička zpráv SMB 2

Každá zpráva začíná hlavičkou pevné velikosti, která obsahuje řídicí příkaz. Délka samotné zprávy je určena řídicím kódem obsaženým v hlavičce a odesílatelem zprávy. Zpráva odeslaná klientem je až na výjimky označována jako požadavek (*request*), zatímco zpráva zasláná serverem je považována za odpověď (*response*).

Pro formát jednotlivých polí existují přesně definované požadavky, které, pokud neexistuje výjimka, musí být dodrženy. Pro textová pole je to především nutnost dodržet formát Unicode verze 5.0, delší záznamy musí dodržet pořadí „Little-endian”, číselné hodnoty musí obsahovat hodnotu z datového typu „integer” s danou délkou a pole označená jako „nepoužitá” musí nést hodnotu nula (Microsoft, 2017, s. 29).

Důležitým polem v hlavičce je položka Flags určující samotný tvar hlavičky. Ta totiž může nabývat dvou tvarů (ASYNC a SYNC) v závislosti na příkazu obsaženém v jejím poli Flags (SMB2_FLAGS_(A)SYNC_COMMAND). Tvar ASYNC je použit pro odpovědi na požadavky, které server zpracovává asynchronně. Nečeká tedy na jejich výsledek a před jejich doručením začne zpracovávat jiné požadavky. Tento tvar může být použit pro všechny zprávy, nicméně musí být použit pro „SMB2 CANCEL Request”.

Tím klient ruší předchozí požadavek, na který dříve obdržel tzv „prozatímní” odpověď (interim message) indikující asynchronní zpracování požadavku serverem (Microsoft, 2017, s. 30-36).

Tabulka 3: SMB 2 Message Header
0...7 8...15

ProtocolId	
StructureSize	CreditCharge
(ChannelSequence/Reserved)/Status	
Command	CreditRequest/Response
Flags	
NextCommand	
MessageId	
...	
AsyncId (ASYNC) / Reserved (SYNC)	
... (ASYNC) / TreeId (SYNC)	
SessionId	
...	
Signature	
...	
...	
...	

- *ProtocolId* identifikuje protokol a musí nést hodnotu 0xFE, 'S', 'M', 'B'.
- *StructureSize* nese vždy hodnotu 64 (bajtů) a udává velikost hlavičky.
- *CreditCharge* je v dialektu SMB 2.0.2 nepoužit a nese hodnotu nula. V ostatních dialektech jsou „kredity” mechanismem k limitování množství požadavků, kterým může klient zatížit server. Pokud klient nemá dostatek kreditů pro provedení požadavku, musí si od serveru vyžádat jejich navýšení.
- *(ChannelSequence/Reserved)/Status* (4 bytes) nese různé hodnoty v každém dialektu. V SMB 3.x požadavku značí ChannelSequence (2 bajty) změnu komunikačního kanálu a Reserved (2 bajty) je nepoužit, v odpovědi slouží u všech dialektů k přenosu chybových zpráv.
- *Command* je řídicí příkaz dané zprávy, musí nést hodnotu jednoho z 19 v SMB 2 platných příkazů.
- *CreditRequest/CreditResponse* slouží k vyžádání a zaslání požadovaných kreditů nutných k provádění dalších příkazů.

- *Flags* ovlivňuje způsob, jakým budou zpracována ostatní pole. Musí nabývat jedné ze sedmi platných hodnot.
- *NextCommand* v jednoduché zprávě nese hodnotu nula, ve složené nese offset k následující SMB 2 hlavičce.
- *MessageId* je unikátní identifikátor zprávy v rámci daného spojení.
- *AsyncId* je unikátní identifikátor přiřazený serverem asynchroně zpracovávanému požadavku.
- *SessionId* je unikátní identifikátor daného spojení. Během jeho navazování nese hodnotu nula.
- *Signature* slouží jako podpis zprávy v případě, že „Flags” obsahuje hodnotu SMB2_FLAGS_SIGNED, jinak je nula.
- *Reserved* nese hodnotu nula, server toto pole ignoruje.
- *TreeId* slouží k identifikaci sdíleného zdroje, ke kterému je vznešen požadavek.

1.2.3 Navázání komunikace se SMB serverem

Klientská stanice zahajuje vyjednávání se serverem zasláním zprávy označované jako „NEGOTIATE Request”. V rámci tohoto požadavku klient zasílá seznam dialektů, kterým klientská implementace SMB2 rozumí. NEGOTIATE Request obsahuje pole „Dialects” proměnné délky, které musí vždy obsahovat alespoň jeden dialekt podporovaný klientskou aplikací (Microsoft, 2017, s. 41).

Serverová aplikace odpovídá na požadavek zprávou „NEGOTIATE Response”. Ta, mimo jiné, obsahuje pole „DialectRevision” o délce 2 bajty. V tomto poli server zasílá informaci o tom, který dialekt SMB2 protokolu server preferuje k další komunikaci. K vyjednávání o dialektu typicky postačuje zaslání jednoho požadavku a odpovídající odpovědi (Microsoft, 2017, s. 45).



Obrázek 2: Navázání SMB2 spojení

Tabulka 4: Hodnoty dialektů pro NEGOTIATE Request/Response

0x0202	Dialekt SMB 2.0.2.
0x0210	SMB 2.1. Nepodporováno ve Windows Vista SP1 a Server 2008.
0x0300	SMB 3.0.0. Stejně jako výše, k tomu Windows 7 a Server 2008 R2.
0x0302	SMB 3.0.2. Stejně jako výše, k tomu Windows 8 a Server 2012.
0x0311	SMB 3.1.1. Stejně jako výše, k tomu Windows 8.1 a Server 2012 R2.
0x02FF	SMB 2.???. Hodnota značí „dvoufázové” vyjednávání dialektu.

Pokud klient podporuje dialekt SMB 2.1 nebo jednu či více z budoucích revizí, může v NEGOTIATE Request zaslat hodnotu 0x2FF. Server poté nezasílá hodnotu preferovaného dialektu, namísto toho odpovídá stejnou hodnotou. Vyjednávání poté pokračuje zasláním dalšího požadavku, v němž klient zasílá seznam podporovaných revizí SMB 2.1 a novějších. Server tentokrát opět odpovídá hodnotou preferovaného dialektu, čímž vyjednávání končí.

Výše popsané vyjednávání se označuje jako tzv. „Multi-Protocol Negotiation”. Pokud klient namísto seznamu podporovaných dialektů zašle pouze hodnotu 0x0202, tedy zahájí vyjednávání dialektu SMB 2.0.2, pak mluvíme o „SMB2 Negotiation” (Microsoft, 2017, s. 46). Po navázání transportního spojení klient pokračuje zasláním zprávy „SESSION_SETUP Request”, čímž se pokouší navázat komunikační spojení se serverem.

SESSION_SETUP Request obsahuje množství rezervovaných bloků, jimž klient nastavuje hodnotu nula. Tato pole server musí ignorovat. Zpráva dále nese tzv. autentizační buffer, v němž je přenášen autentizační token typicky vytvořený protokolem GSS neboli programovým rozhraním „Generic Security Services” (Microsoft, 2017, s. 48-50).

Významnou součástí této zprávy je dvoubajtové pole „PreviousSessionId”. To slouží ke znovunavázání spojení v případě krátkodobého výpadku sítě. Server pomocí této hodnoty dokáže identifikovat chybou zasažené předchozí spojení.

Tabulka 5: SMB 2 SESSION_SETUP Response

0...7	8...15	16...23	24...31
StructureSize		SessionFlags	
SecurityBufferOffset		SecurityBufferLength	
Buffer			
...			

Server odpovídá krátkou zprávou „SESSION_SETUP Response”, jejímž účelem je nést buffer obsahující autentizační token. Pole *SecurityBufferOffset* udává vzdálenost autentizačního bufferu od začátku transportní SMB2 hlavičky. Délka bufferu je obsažena v poli *SecurityBufferLength*. Pokud autentizaci inicioval server, pak je token vytvořen rozhraním GSS. Inicioval-li autentizaci klient, pak tento klient disponuje možností zvolit autentizační protokol. Pole *StructureSize* udává velikost pevné části zprávy, tedy bez proměnné části autentizačního bufferu. *SessionFlags* může nést např. informaci o tom, zda byl klient autentizován jako host nebo anonymní uživatel (Microsoft, 2017, s. 50-51).

Pokud bylo při navazování spojení vyjednáno rozšířené zabezpečení, může server při zpracování tokenu obdržet chybu STATUS_MORE_PROCESSING_REQUIRED, kterou zasílá v odpovědi příslušnému klientovi. Klient po obdržení této chyby musí dodat další autentizační údaje, což vyústí v zaslání dalšího požadavku o navázání spojení. Pokud jsou dodány požadované údaje a autentizační token je korektně zpracován, odpovídá server zprávou se stavem STATUS_SUCCESS. Komunikační spojení tak bylo navázáno (Microsoft, 2017, s. 197-198).

Po úspěšné autentizaci klientu zbývá zaslat požadavek pro připojení ke sdílenému zdroji (share). K tomu slouží zpráva „SMB2 TREE_CONNECT Request” nesoucí buffer obsahující název sdíleného zdroje, ke kterému chce klient přistupovat.

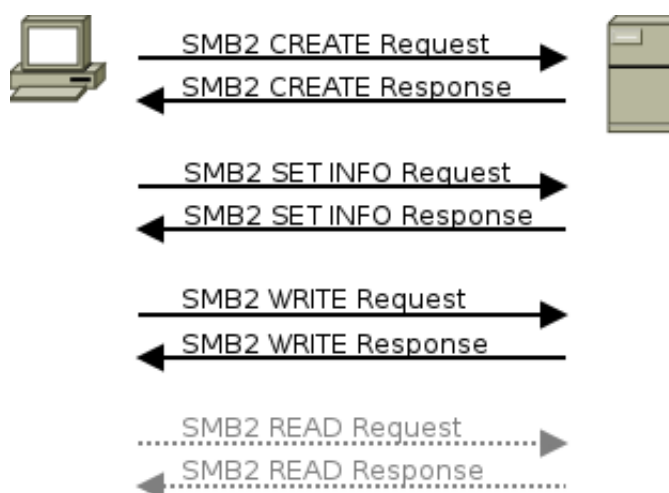
Pokud je požadavek úspěšně zpracován, server zasílá zprávu „SMB2 TREE_CONNECT Response” nesoucí mimojiné typ sdíleného zdroje (Microsoft, 2017, s. 52-55).

Tabulka 6: Typy sdílených zdrojů

SMB2_SHARE_TYPE_DISK 0x01	vzdálené umístění
SMB2_SHARE_TYPE_PIPE 0x02	vzdálená roura (IPC\$)
SMB2_SHARE_TYPE_PRINT 0x03	vzdálená tiskárna (PRINT\$)

1.2.4 Čtení a zápis souborů pomocí SMB 2

Pro otevření souboru klient zasílá zprávu „SMB2 CREATE Request”. Ta, pokud je ignorována proměnná délka bufferu, nabývá pevné délky 57 bajtů. Tato zpráva nese v k tomu určených polích informaci o typu sdíleného zdroje i jeho attributech. Tento požadavek o „vytvoření” je používán jak pro čtení a zápis souborů, tak pro připojení k IPC\$ nebo sdíleným tiskárnám (Microsoft, 2017, s. 56).



Obrázek 3: Zápis a čtení souboru

Jedním z klíčových polí této zprávy je pole „ShareAccess” o velikosti 4 bajty. Při zpracování „běžných” souborů toto pole nabývá jedné ze tří hodnot dle vyžadované operace (čtení, zápis, vymazání). Žádná z těchto hodnot by neměla být nastavena, pokud klient žádá o přístup k pojmenované rouře nebo tiskárně. Server tato pole v těchto případech pro jistotu ignoruje. Požadavek o přístup k tiskárně vždy vede k vytvoření nového souboru, v případě pojmenované roury je výsledkem její nová instance (Microsoft, 2017, s. 57-58).

Tabulka 7: Hodnoty pole ShareAccess

FILE_SHARE_READ (0x00000001)	Čtení souboru.
FILE_SHARE_WRITE (0x00000002)	Zápis do souboru.
FILE_SHARE_DELETE (0x00000004)	Smazání souboru.

Pole „CreateDisposition” (4 bajty) definuje akci serveru v případě, že požadovaný soubor již existuje. V požadavku o pojmenovanou rouru pole může nést libovolnou hodnotu, server ji ale musí ignorovat (Microsoft, 2017, s. 58).

Tabulka 8: Hodnoty pole CreateDisposition

FILE_SUPERSEDE 0x00000000	Nahraď nebo vytvoř nový soubor. Hodnota by neměla být použita pro objekt tiskárny.
FILE_OPEN 0x00000001	Požadavek o prosté otevření, vyhodnoceno jako úspěch. Nesmí být použito na objekt tiskárny.
FILE_CREATE 0x00000002	Požadavek o vytvoření nového souboru. Pro existující soubor vyhodnoceno jako neúspěch.
FILE_OPEN_IF 0x00000003	Otevři případně vytvoř nový soubor. Hodnota by neměla být použita pro objekt tiskárny.
FILE_OVERWRITE 0x00000004	Přepiš existující soubor. Nesmí být použito na objekt tiskárny.
FILE_OVERWRITE_IF 0x00000005	Přepiš pouze pokud soubor skutečně existuje. Hodnota by neměla být použita pro objekt tiskárny.

Dalším důležitým polem je pole „CreateOptions” (4 bajty) specifikující operace prováděné serverem při otevírání nebo vytváření souborů. Toto pole musí nést alespoň jednu z definovaných hodnot, nebo kombinaci některých z nich. Možné kombinace jsou určeny především pozicí bitů příslušných hodnot v tomto poli. Hodnoty tohoto pole jsou nepřímo určeny hodnotou v poli CreateDisposition. Na neplatnou kombinaci těchto dvou polí, případně na neplatnou kombinaci hodnot v poli CreateOptions server reaguje zasláním zprávy „SMB2 CREATE Response” nesoucí ve „flags” hodnotu „STATUS_INVALID_PARAMETER” (Microsoft, 2017, s. 58).

Řada hodnot tohoto pole by měla nést hodnotu nula a server by je měl zcela ignorovat. Těmito hodnotami jsou například FILE_SYNCHRONOUS_IO_ALERT, FILE_OPEN_REMOTE_INSTANCE nebo FILE_OPEN_BY_FILE_ID, na jehož „nastavení” server reaguje zprávou nesoucí status „STATUS_NOT_SUPPORTED”.

Důvodem tohoto požadavku jsou SMB 2 klienti z rodiny operačních systémů MS Windows, kteří tyto hodnoty nevyužívají. Ostatní hodnoty v tomto poli slouží k řešení konfliktů jmen

mezi adresáři a soubory, k nastavení okamžitého zápisu souboru do perzistentní paměti, sekvenčnímu nebo náhodnému přístupu do souboru nebo například k vymazání souboru po zpracování všech požadavků o jeho otevření (Microsoft, 2017, s. 58-60).

Po úspěšném zpracování požadavku na otevření server odpovídá zprávou „SMB2 CREATE Response” s identifikátorem otevřeného souboru. Tato zpráva ve své pevné části nabývá délky 89 bajtů. Klient je touto zprávou informován o průběhu otevření souboru. Tato informace je daná hodnotou pole „CreateAction” s délkou 4 bajty. Zpráva obsahuje i řadu časových údajů udávajících datum vytvoření, modifikace či posledního čtení a zápisu. Obsažena je i velikost souboru, jeho atributy a další informace popisující jednotlivé soubory (Microsoft, 2017, s. 73-76).

Tabulka 9: Hodnoty pole CreateAction

FILE_SUPERSEDED (0x00000000)	Existující soubor byl smazán a vytvořen.
FILE_OPENED (0x00000001)	Existující soubor byl otevřen.
FILE_CREATED (0x00000002)	Byl vytvořen nový soubor.
FILE_OVERWRITTEN (0x00000003)	Existující soubor byl přepsán.

Pro doplnění informací o souboru klient zasílá zprávu „SMB2 SET_INFO Request”. Druh informací zasílaných klientem udává pole „InfoType” o délce jednoho bajtu. Doplnková data o souboru jsou rozdělena do čtyř kategorií. Jsou to data popisující soubor, souborový systém, zabezpečení nebo kvótu souboru. Na tuto zprávu server reaguje krátkou odpovědí. Ta nenese žádnou informaci, pouze údaj o své velikosti. Slouží jako pouhé potvrzení, jinak nedisponuje žádnou informační hodnotou (Microsoft, 2017, s. 126-128).

Tabulka 10: Zpráva SMB2 SET_INFO Response

0...7	8...15
StructureSize	

Zápis dat do souboru klient provádí odesláním zprávy „SMB2 WRITE Request” na níž server reaguje zprávou „SMB2 WRITE Response”. Při zápisu většího množství dat dochází k výměně těchto zpráv mezi klientem a serverem opakovaně. Každá zpráva zaslaná klientem obsahuje identifikátor cílového souboru, zapisovaná data, jejich délku a množství dat, které zbývá zaslat. V dialektech z rodiny SMB 3.x zpráva nese i informaci o příslušném komunikačním kanálu.

Podpora více kanálů v dialektové rodině SMB 3.x umožňuje rovnoměrně rozložit zátěž na více kanálů v rámci jednoho spojení, což umožňuje zvýšit datovou propustnost při nižším systémovém zatížení (Microsoft, 2017, s. 88-90).

- *StructureSize* udává pevnou velikost zprávy, tedy bez proměnné části bufferu. Nese vždy hodnotu 2 (bajty).
- *DataOffset* je vzdáleností začátku odesílaných dat od začátku SMB 2 hlavičky.
- *Length* určuje velikost odesílaných dat.
- *Offset* je pozicí v cílovém souboru, kam mají být odesílání data zapsána.
- *FileId* představuje SMB 2 FileID strukturu, která je identifikátorem cílového souboru.
- *Channel* je v dialektech SMB 2.x rezervovaným polem a musí být ignorován. V Dialektech SMB 3.0 identifikuje příslušný kanál, přes který probíhá komunikace.
- *RemainingBytes* jsou klientem dosud nezaslaná data. Určují kolik dat ještě server obdrží.
- *WriteChannelInfoOffset* a *WriteChannelInfoLength* jsou v dialektech SMB 2.0.2 a 2.1 ignorovány a nesou hodnotu nula. V dialektu SMB 3.0 nesou informace o pozici a délce dat pro příslušný kanál.
- *Flags* nabývá hodnot
 - *SMB2_WRITEFLAG_WRITE_THROUGH* pro zápis do perzistentní paměti před odesláním odpovědi nebo
 - *SMB2_WRITEFLAG_WRITE_UNBUFFERED* pro blokování cachování dat ve vrstvách nad perzistentní pamětí.
 - První z těchto hodnot není podporována v dialektu SMB 2.0.2, druhá není podporována ani v dialektech SMB 2.1 a 3.0.
- *Buffer* obsahuje samotná odesílaná data.

Server zpracování zaslaných dat oznamuje krátkou zprávou „SMB2 WRITE Response” (Microsoft, 2017, s. 90-91).

Tabulka 11: SMB 2 WRITE Request

StructureSize		DataOffset	
Length			
Offset			
...			
FileId			
...			
...			
...			
Channel			
RemainingBytes			
WriteChannelInfoOffset		WriteChannelInfoLength	
Flags			
Buffer			
...			

Tabulka 12: SMB 2 WRITE Response

StructureSize		Reserved	
Count			
Remaining			
WriteChannelInfoOffset		WriteChannelInfoLength	

- *StructureSize* udávající velikost zprávy nese vždy hodnotu 17 bajtů.
- *Reserved* je rezervované pole.
- *Count* oznamuje množství dat, které server obdržel při předchozím požadavku.
- *Remaining* je rezervované pole.
- *WriteChannelInfoOffset* a *WriteChannelInfoLength* jsou taktéž rezervovaná pole. Server jejich hodnotu nastavuje na nula a klient je musí ignorovat.

Pro opačnou operaci, tedy pro čtení sdíleného souboru, klient zasílá zprávu „SMB2 READ Request”. Ta má velice podobnou strukturu jako požadavek k zápisu dat, odlišuje ji především pole „MinimumCount” o velikosti 4 bajtů. Klient v tomto poli zasílá minimální množství dat, které požaduje zaslat v odpovědi. Server musí odpovědět chybovým stavem, pokud nedokáže přečíst alespoň toto požadované množství. Přečtená data server odesílá v bufferu zprávy „SMB2 READ Response” (Microsoft, 2017, s. 85-88).

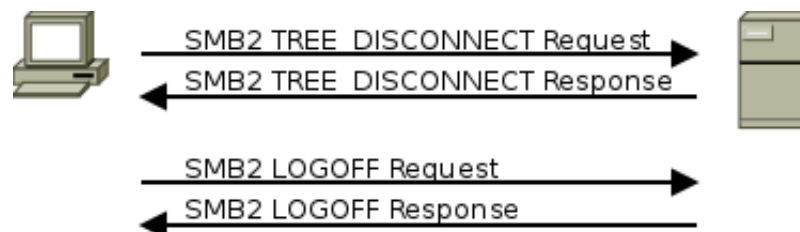
Tabulka 13: SMB 2 READ Response
0...7 8...15 16...23 24...31

StructureSize	DataOffset	Reserved
DataLength		
DataRemaining		
Reserved		
Buffer		
...		

- *StructureSize* udává pevnou velikost zprávy bez proměnné části bufferu. Nese vždy hodnotu 17 bajtů.
- *DataOffset* značí vzdálenost od začátku SMB2 hlavičky k odesílaným datům.
- *Reserved* jsou rezervovaná pole. Server je nuluje a klient ignoruje.
- *DataLength* oznamuje množství dat, které server touto zprávou odesílá.
- *DataRemaining* popisuje množství dat odesílaných kanálem specifikovaným v požadavku čtení.
- *Buffer* proměnné délky obsahuje odesílaná data.

1.2.5 Odpojení se od sdíleného zdroje

Pro ukončení komunikace klientské stanice se SMB2 serverem je nutné provést dvě vzájemné výměny zpráv. Účelem první výměny je odpojení klienta od datové struktury, potažmo od sdíleného zdroje. Po odpojení klient pokračuje požadavkem o odhlášení, čímž žádá o ukončení aktivního spojení (Session). Protože tento proces nevyžaduje žádné dodatečné informace, jedná se o velice malé datové struktury (Microsoft, 2017, s. 55).



Obrázek 4: Odpojení sdíleného zdroje

Klient nejdříve zasílá žádost o odpojení sdíleného umístění, respektive stromové struktury souborů. Identifikátor `TreeId` je obsažen již v SMB2 hlavičce, v těle zprávy „SMB2 TREE_DISCONNECT Request” by tedy byl duplicitní. Server odpovídá prakticky identickou zprávou, klientem vyhodnocenou jako „SMB2 TREE_DISCONNECT Response” (Microsoft, 2017, s. 56).

Odpojení klientské aplikace pokračuje požadavkem „SMB2 LOGOFF Request”. Touto zprávou klient žádá o ukončení aktivního spojení (Session), jehož identifikátor je opět obsažen již v SMB2 hlavičce. Tělo zprávy je znovu prakticky prázdné. Server odpovídá zprávou „SMB2 LOGOFF Response”, čímž vzájemná interakce končí (Microsoft, 2017, s. 51).

Tabulka 14: TREE_DISCONNECT a LOGOFF zprávy

0...7	8...15	16...23	24...31
StructureSize	Reserved		

2 Network File System

NFS neboli „síťový souborový systém” byl vytvořen v roce 1984 ve společnosti Sun Microsystems s cílem vytvořit jednoduše portovatelný nástroj pro přístup ke vzdáleným souborovým systémům. Pro co největší jednoduchost byl NFS navržen jako bezstavový protokol pracující nad balíkem procedur RPC a standardem XDR pro vnější reprezentaci dat. Díky své transparentnosti NFS umožňuje aplikacím přistupovat ke vzdáleným souborům stejným způsobem, jako přistupují k souborům lokálním. Díky tomu nebylo třeba aplikace jakkoli upravovat pro mapování vzdálených úložišť (Sandberg, 1985, s. 1).

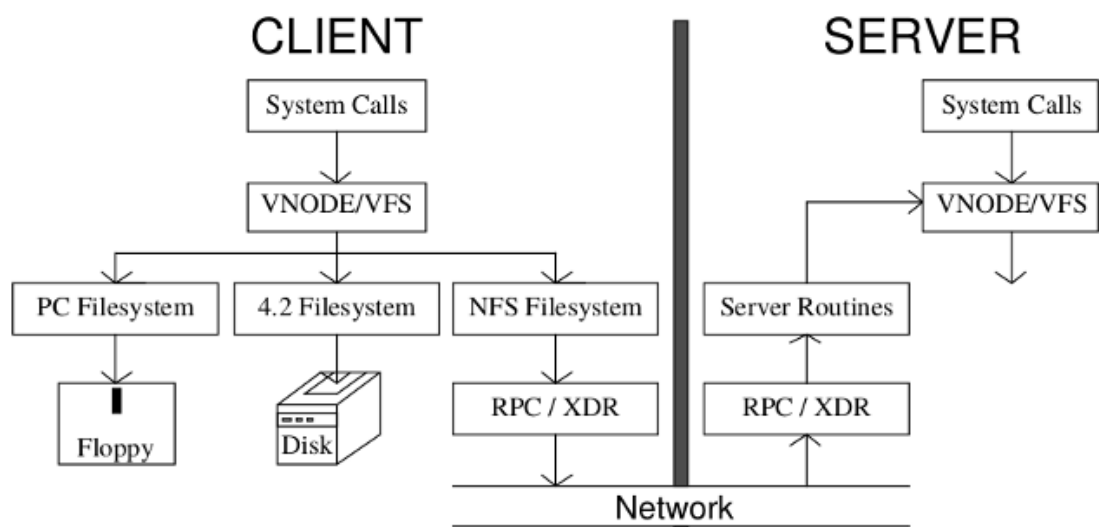
K dosažení této transparentnosti bylo nutno přidat do tehdejšího unixového jádra mezivrstvy zvané VFS/vnode. Dvojice „virtual file system” a „virtual node” sloužila jako rozhraní mezi skutečným souborovým systémem a nad ním pracujícími aplikacemi. Zatímco VFS definovalo operace nad celým souborovým systémem, virtuální uzel definoval operace nad uzly, tedy soubory a adresáři v rámci daného souborového systému. Účelem této mezivrstvy bylo umožnit implementaci dalších souborových systémů pod univerzálním rozhraním, bez nutnosti dále zasahovat do stávajících implementací (Sandberg, 1985, s. 4-5).

Přidání mezivrstvy mezi souborový systém a aplikace hrozilo snížením výkonu. Měření provedená v té době vývojáři Sun Microsystems ale ukázala, že zpomalení způsobené rozhraním VFS/vnode bylo sotva měřitelné a v nejhorších případech dosahovalo pouze 2 % původních hodnot (Sandberg, 1985, s. 5).

Kromě transparentního přístupu k souborům bylo cílem vzniku NFS vytvořit na stroji a operačním systému nezávislý nástroj pro sdílení souborů. Přestože vývoj NFS probíhal na systému Unix, je na něm zcela nezávislý, což dokázalo jeho značné rozšíření. Dalším cílem bylo vytvořit nástroj kvalitně řešící obnovení po selhání. Z tohoto důvodu byl protokol navržen jako bezstavový. Problém obnovení po selhání např. síťového připojení tím téměř zcela odpadá (Sandberg, 1985, s. 2).

NFS užívá k transportu svých zpráv datagramový protokol UDP. Ten je již ze své podstaty bezstavový, případné udržování spojení tak musí být implementováno přímo v aplikaci. NFS servery ale žádnou takovou správu spojení nevykonávají. Zprávy zasílané v rámci NFS v sobě vždy nesou všechny informace potřebné ke kompletnímu zpracování požadavku.

Dojde-li tedy k pádu serveru, jediným mechanismem „obnovení“ je opakované zasílání dané zprávy klientem a to až do chvíle úspěšného doručení odpovědi. Alternativou je nastavení časovače, po jehož vypršení klient ohlásí chybu. Pokud ale není použit časovač, klient prakticky nerozpozná rozdíl mezi serverem pomalým a serverem, u něhož došlo k selhání a následnému restartu. Selhání klientské aplikace žádné obnovení stavu nevyžaduje (Sandberg, 1985, s. 2).



Obrázek 5: Princip protokolu NFS (Sandberg, 1985, s. 4)

2.1 Remote Procedure Call

Pro přenos zpráv byl protokol NFS vystavěn nad na operačním systému nezávislé technologii „vzdáleného volání procedur” neboli RPC. Použití procedur zjednodušuje tvorbu síťových aplikací oddělením síťové části od aplikační logiky programu. Operace čtení a zápisu z a do soketu jsou nahrazeny voláním procedury s definovanými parametry, přičemž odpověď protistrany je obsažena v návratové hodnotě (Sun, 1988, s. 2-3).

Výhodou RPC je nezávislost na protokolu transportní vrstvy. Ačkoli tedy NFS již od svého vytvoření používá k přenosu dat datagramový protokol UDP, je možné ho nahradit libovolným protokolem transportní vrstvy bez zásahu do vyšších vrstev. Při dodržení rozhraní RPC nemusí aplikační logika projít žádnou revizí.

Formát dat přenášených po síti je definován standardem External Data Representation (XDR). Tento standard definuje strojově nezávislý formát jak základních, tak i složených datových typů. Popisuje například velikost základního bloku a pořadí bajtů jednotlivých datových typů. XDR obsahuje i vlastní „jazyk”, nejedná se ale o jazyk programovací. Jeho úkolem je popsat formát argumentů a návratových hodnot RPC procedur (Srinivasan, 1995, s. 2).

Tabulka 15: XDR reprezentace typu integer

0...7	8...15	16...23	24...31
bajt 0	bajt 1	bajt 2	bajt 3
MSB		LSB	

2.1.1 Zprávy RPC

Pro RPC komunikaci jsou užívány dva typy zpráv, přičemž oba začínají dvěma společnými poli. Prvním je pole „xid” o délce 32 bitů. Toto pole slouží jako transakční identifikátor jednotlivých požadavků či volání. Identifikátor odpovědi vždy odpovídá identifikátoru příslušného volání. Druhou společnou položkou je enumerátor (Sun, 1988, s. 9). Ten určuje, zda je zpráva voláním nebo odpovědí.

Pro volání vzdálených procedur slouží zpráva „volání” neboli RPC Call. Tato zpráva obsahuje jak všechny údaje nutné k identifikaci správné verze RPC mechanismu, tak i k nalezení správné procedury (Sun, 1988, s. 9-10).

Algoritmus 1 Struktura RPC zprávy (Sun, 1988, s. 9)

```
struct rpc_msg {  
    unsigned int xid;  
    union switch (msg_type mtype) {  
        case CALL: call_body cbody;  
        case REPLY: reply_body rbody;  
    } body;  
};
```

Tabulka 16: RPC Call and Reply messages (Maiti, 2011, s. 22-29)

0...7	8...15	16...23	24...31	0...7	8...15	16...23	24...31
IP header (20 bytes)				IP header (20 bytes)			
UDP header (8 bytes)				UDP header (8 bytes)			
Identifier XID				Identifier XID			
Call enum (0)				Reply enum (1)			
RPC version				Status			
Program number				Verifier (up to 400 bytes)			
Version number				Accept status			
Procedure number				Procedure results (variable)			
Credentials (up to 408 bytes)							
Verifier (up to 408 bytes)							
Procedure call attributes (variable)							

- *RPC version* značí verzi použitého RPC protokolu a dle platného RFC musí nést hodnotu 2.
- *Program number* slouží jako identifikátor volajícího programu. Tato čísla spravuje a jednotlivým programům přiřazuje centrální autorita. Tou byla v minulosti společnost Sun Microsystems, dnes je jí organizace IANA.
- *Version number* udává verzi daného programu. Slouží k odlišení jednotlivých revizí klientských aplikací a protokolů používajících RPC.
- *Procedure number* specifikuje proceduru volaného programu, které budou předány atributy obsažené v dané zprávě.
- *Credentials* slouží k autorizaci klienta. Nese informaci o tom, jestli má klient oprávnění ke komunikaci se vzdáleným programem.

- *Verifier* slouží k autentizaci klienta. Jednoznačně identifikuje volající program.
- *Procedure call attributes* je pole proměnné délky obsahující data určené ke zpracování volanou procedurou.
- *Status* informuje klienta o tom, zda bylo jeho volání přijato nebo odmítnuto.
- *Accept status* závisí na hodnotě předchozího pole *Status*.
 - Pokud bylo volání přijato, pak toto pole informuje o úspěšnosti vykonání procedury popřípadě o její nedostupnosti či nedostupnosti volaného programu.
 - Pokud bylo volání odmítnuto, informuje zde server o chybné verzi RPC protokolu nebo chybě autentizace.

2.1.2 Mapování portů

Pro úspěšnou síťovou komunikaci musí klientská aplikace znát port, na němž vzdálený program naslouchá. Porty jsou rozděleny do tří skupin podle jejich využití a „důležitosti“, přičemž většina z nich je spravována centrální autoritou, kterou je organizace „Internet Assigned Numbers Authority“ (IANA). Ta jednotlivé porty přiřazuje aplikacím a službám pevně či na základě požadavku jejich autorů (Reynolds, 1992, s. 2).

Tabulka 17: Rozdělení síťových portů

0 - 1023	Well known (System) ports
1024 - 49151	Registered (User) ports
49152 - 65535	Dynamic (Private) ports

Well known ports

Dobře známé porty nebo také systémové porty slouží pro systémové služby poskytující široce užívané síťové služby. Na unixových operačních systémech na těchto portech může naslouchat pouze služba spuštěná s oprávněním administrátora (root). Účelem tohoto omezení je důvěryhodnost služby. Uživatelé se tedy ke službám na těchto portech mohou připojovat bez obavy podvrhnutí služby jiným uživatelem (Reynolds, 1992, s. 9).

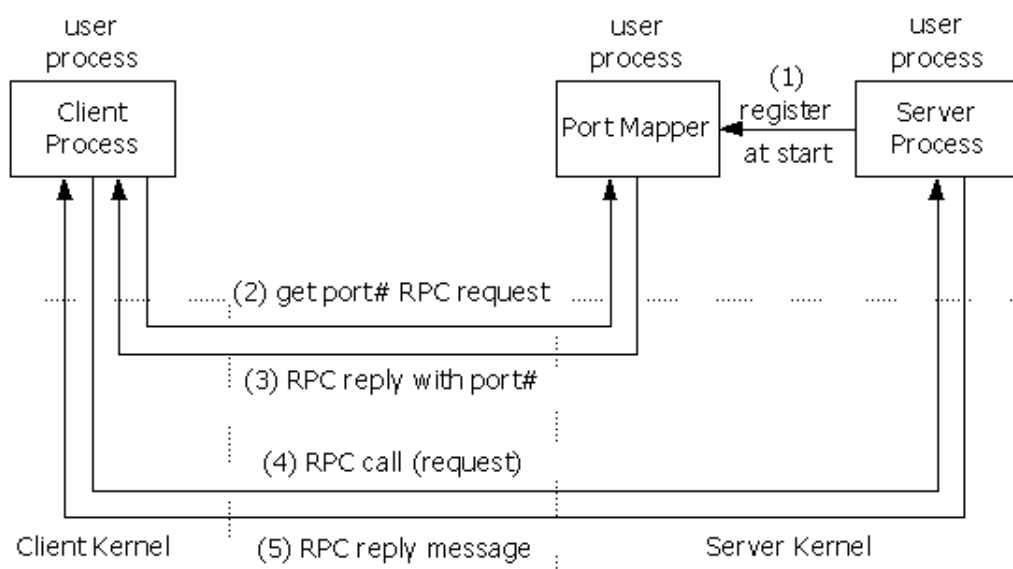
Registered ports

Pro přiřazení registrovaného portu by měl autor služby podat oficiální žádost organizaci IANA. Jedná se o uživatelské porty, typicky nevyžadující speciální oprávnění administrátora (Reynolds, 1992, s. 23).

Dynamic ports

Dynamické či „dočasné“ (ephemeral) porty slouží k dynamické alokaci portu pro klientské aplikace. Každá klientská aplikace komunikující se síťovými službami běžícími na portech z předešlých skupin taktéž potřebuje port. Ten je jí náhodně přiřazen ze skupiny dynamických portů, přičemž toto přiřazení nemívá dlouhého trvání. Při každém spuštění může být klientské aplikaci přiřazen jiný port.

Rozšířenou praxí je nicméně neoficiální užívání dobře známých i registrovaných portů aplikacemi a službami, které pro to nemají oprávnění. Výjimkou není ani více aplikací užívajících stejný port. Například port 465 široce užívaný autentizovaným SMTP (Simple Mail Transfer Protocol) je organizací IANA oficiálně vyhrazen pro protokol URD (URL Rendezvous Directory) (Touch, 2017, s. 9).



Obrázek 6: Portmap principle (Maiti, 2011, s. 22)

Programům využívajícím vzdálené volání procedur je port přiřazován dynamicky při jejich spuštění. Za toto přiřazování je zodpovědný tzv. Port Mapper, známý také jako portmap nebo rpcbind.

Hlavní výhoda užití portmapu je síťově transparentní alokace portů jednotlivým programům. Množství RPC implementujících programů může být teoreticky větší, než množství rezervovaných portů jak na jednotlivých stanicích, tak napříč sítí (Sun, 1988, s. 23-24).

Jednotlivé programy se při svém spuštění „registrují“ u lokálního portmapu, který jim přiřadí některý z aktuálně volných portů, čímž vytváří „mapu portů“. Výsledkem může být, že identický program naslouchá na každé stanici v síti na jiném portu. Pokud chce klientský program kontaktovat server, dotáže se nejdříve Portmapu daného serveru, který jako jediný vždy naslouchá na pevně přiřazeném portu 111.

Portmap na základě RPC požadavku prohledá svou mapu portů. Najde-li shodu pro požadovaná čísla programu a procedury, zasílá zpět odpovídající číslo portu. Klient poté zasílá druhý požadavek přímo cílovému programu (Sun, 1988, s. 24).

Algoritmus 2 Portmap entry structure (Sun, 1988, s. 22)

```
struct mapping {  
    unsigned int prog;    // Program, nfs for example  
    unsigned int vers;    // Program version  
    unsigned int prot;    // Protocol tcp/udp  
    unsigned int port;    // Port number  
};
```

Portmap řeší i problém všesměrových RPC zpráv. Pokud cílový program naslouchá na různých portech napříč sítí, nelze jednoduše zaslat všesměrovou zprávu jedním voláním na neexistující společný port. Namísto toho je zpráva opět zaslána na port 111, tedy všem instancím portmapu v síti. Ten posléze informuje příslušný program (Sun, 1988, s. 7).

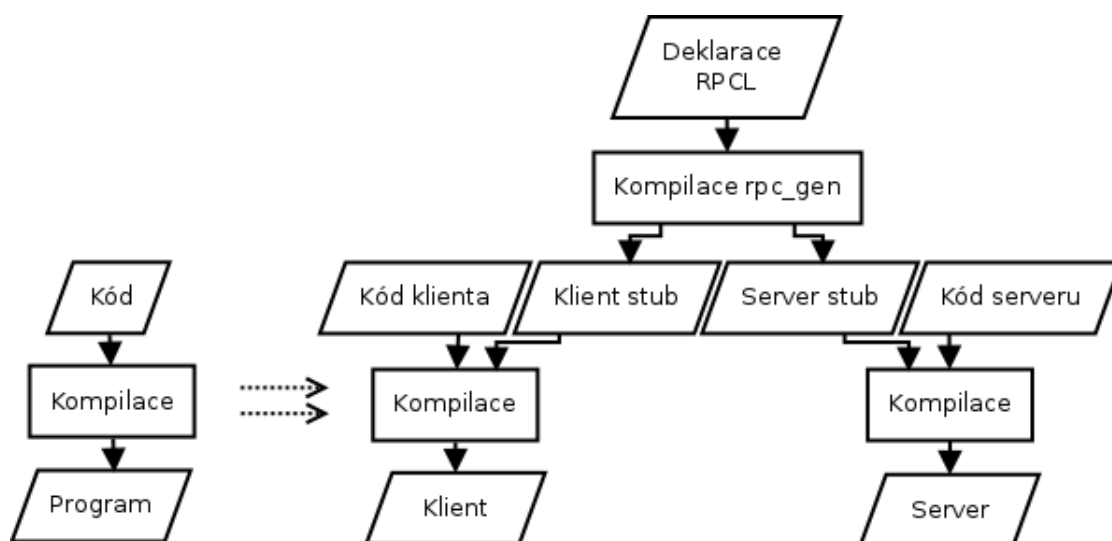
Princip vzdáleného volání procedur

Cílem vzdáleného volání procedur je stejně jako u procedur lokálních zjednodušení programu. V tomto případě nejenom odstranění duplicitního kódu a jeho přesunutí do samostatné procedury, ale především odstranění síťové od aplikační logiky. Pokud se vývojář nemusí soustředit na síťové operace, může věnovat více času hlavní činnosti své aplikace. A právě za tímto účelem vznikl kompilátor `rpc_gen` (Petron, 1997, s. 2).

Jeho úkolem je generování síťových procedur a operací v jazyce C, které vytváří na základě předepsaných parametrů. Cílem programátora je převést lokální funkce jeho aplikace na procedury schopné vykonávat identickou činnost vzdáleně.

Prvním krokem k dosažení tohoto cíle je vytvoření deklarativního souboru v jazyce RPCL (Remote Procedure Call Language). Tento soubor obsahuje pouze deklarace vzdálených procedur, jejichž tvar udávají pouze originální deklarace v jazyce C a jim odpovídající zápis v RPCL. Tento zdrojový soubor (programový kód ve formátu .x) poté zpracuje kompilátor `rpc_gen` a na jeho základě vytvoří hlavičkový soubor se společnými deklaracemi a tzv. „stub” soubory (Petron, 1997, s. 2).

Tyto soubory tvoří mezivrstvu mezi aplikací respektive vlastním kódem vývojáře a RPC knihovnou. Do „stub” souborů tedy `rpc_gen` generuje operace, které přebírají informace a data klientské aplikace a v součinnosti s RPC knihovnou zajišťují vše potřebné k úspěšnému přenosu dat po síti. Na straně serveru dochází ke zpětné transformaci dat knihovnou a skrze serverový „stub” k volání příslušné procedury. Tuto vzdálenou proceduru již musí opět vytvořit vývojář. Vzdálená procedura se téměř neliší od té lokální, výjimkou je její hlavička. Ta musí korespondovat s hlavičkou deklarovanou v .x souboru (Petron, 1997, s. 2-3).



Obrázek 7: Postup kompilace lokálního a RPC programu

Největší úpravy s cílem vytvořit RPC program je nutné provést ve zdrojovém kódu klientské aplikace. O připojení k serveru a volání příslušné procedury rozhoduje vývojář a to pomocí funkcí z klientského „stub” souboru. Jejich použití je velice snadné, protože vše, co od vývojáře požadují, je znát volaný server a proceduru (Petron, 1997, s. 3).

Síťový souborový systém

Nástroj NFS je jedním z mnoha programů stavějících na mechanismu vzdáleného volání procedur. Organizací IANA mu bylo přiřazeno programové číslo 100003 (Shepler, 2015, s.1). Samotný protokol NFS je tvořen sestavami procedur pracujících na úrovních RPC a VFS/vnode. Tyto procedury jsou na sebe většinou přímo mapovány, čímž je omezena režie zpracování dat (Sun, 1988, s. 7). Podrobný popis všech těchto procedur je ale nad rámec této práce.

2.2 NFS verze 4

Implementace první verze Network File System protokolu započala v březnu roku 1984. NFSv1 nicméně sloužil výhradně pro testovací účely a nikdy neopustil laboratoře Sun Microsystems. Po začlenění řady změn do původní specifikace byl protokol vydán pod označením „Network File System version 2” v podobě veřejně přístupného dokumentu. Tímto dokumentem je RFC 1094 vydané v březnu 1989 (Sun, 1989, 1).

K vydání třetí verze protokolu NFS došlo v polovině roku 1995 zveřejněním RFC 1813. NFSv3 se nijak zásadně neodlišoval od svého předchůdce. Jeho účelem bylo řešit především výkonnostní nedostatky původního protokolu. S tím souviselo i zahrnutí transportního protokolu TCP, jehož podpora napříč výrobci síťových zařízení v té době začala vzrůstat (Callaghan, 1995, 1).

2.2.1 Změny v protokolu NFSv3

- Adresní prostor pro velikost souborů vzrostl z 32 na 64 bitů. NFSv2 využíval výhradně 32 bitové adresování souborů a to včetně znaménkového bitu. Velikost souboru se kterým mohl protokol pracovat tak byla limitována na 2 GB. Některé implementace tento problém obcházely použitím znaménkového bitu pro adresu souboru, čímž limit velikosti vzrostl na 4 GB. Limit velikosti souboru určuje lokální souborový systém serveru (Callaghan, 1995, 120).
- Velkou nevýhodou bylo synchronní zpracování požadavků protokolem NFSv2. Klient tak musel před každým požadavkem čekat na odpověď předcházejícího požadavku.

Ačkoli neoficiálně byla podpora asynchronních požadavků zahrnuta do mnoha implementací NFSv2, oficiálně byla zahrnuta až ve třetí verzi protokolu (Callaghan, 1995, 10).

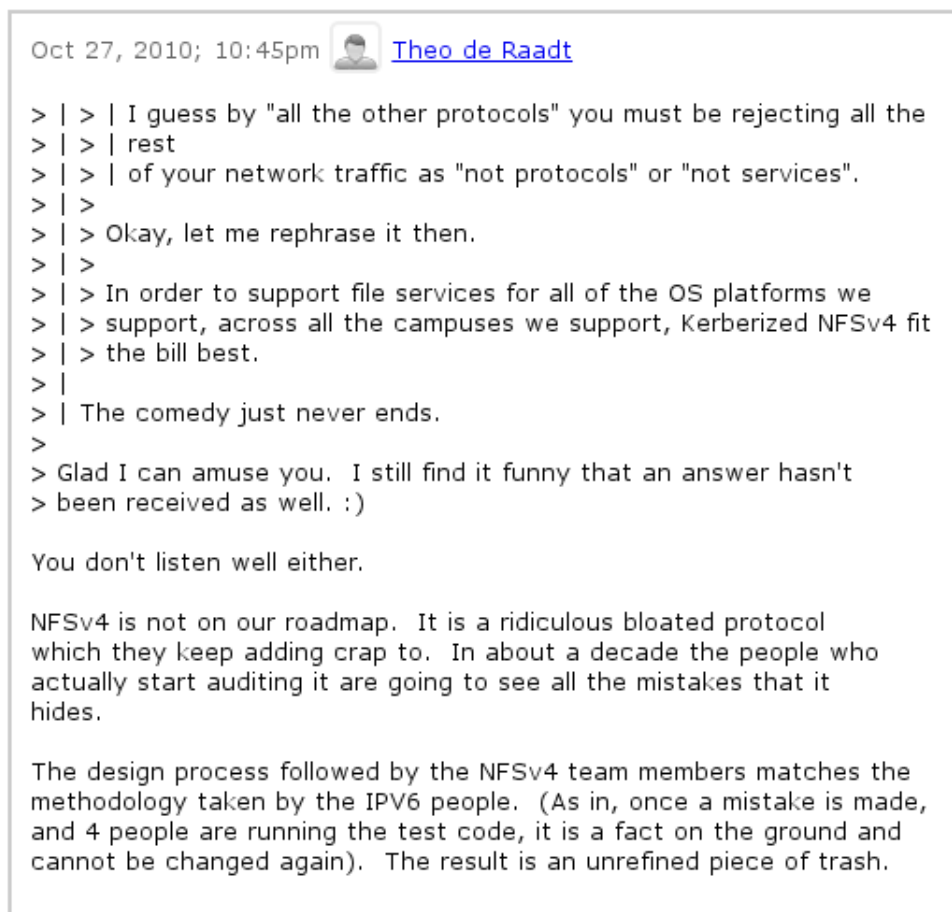
- Pro snížení počtu zpráv odesílaných klienty k serveru bylo rozšířeno množství atributů v řadě požadavků. Klient tak nemusel dodatečné informace „opakovaně“ žádat od serveru, protože je obdržel v rámci jedné odpovědi. Snížilo se tak vytížení sítě i příslušného serveru.
- Podpora TCP umožnila lepší použití protokolu napříč WAN. Zmizel také 8 KB limit uplatňovaný na NFSv2 požadavky. Tento limit byl překonán při použití UDP i TCP jako transportního protokolu (Callaghan, 1995, 14).

Ačkoli je NFSv3 značně rozšířen a je podporován všemi velkými „hráči“ na poli výrobců síťových úložišť, nikdy nebyl prohlášen standardem. Přesto se světle výhod jeho rozšíření napříč celým průmyslem v Sun Microsystems rozhodli vzdát se kontroly nad jeho budoucím vývojem (Callaghan, 1995, 1). Nová verze protokolu tak již vznikala pod záštitou organizace „Internet Engineering Task Force“ (IETF).

2.2.2 Network File System 4.x

NFSv4 bylo vydáno v dubnu 2003 v podobě RFC 3530. Současně se svým vydáním byl protokol prohlášen internetovým standardem. Protokol byl pod vedením IETF kompletně přepracován tak, aby adresoval všechny nedostatky svého předchůdce (Shepler, 2003, s. 8). Ne všichni ale souhlasili s označením určitých vlastností NFSv3 za nedostatky, v důsledku čehož se nové verzi protokolu dostalo velmi chladného přijetí. Jedním z největších kritiků byl ve své době zakladatel a vedoucí představitel projektů OpenBSD a OpenSSH Theodore Raadt. Ten „nový protokol“ označil za směšně nafouklý a plný nesmyslů. Vyjádřil také vážné pochybnosti o procesu jeho vývoje (Raadt, 2010).

Na rozdíl od svých předchůdců byl NFSv4 navržen jako stavový protokol, v důsledku čehož na transportní vrstvě spoléhá výhradně na protokol TCP. Použití UDP u NFSv4 není možné. Protokol adresuje také další z „nedostatků“ NFSv3. Přišel tak o svou modularitu a to s cílem zvýšit výkon především v sítích WAN.



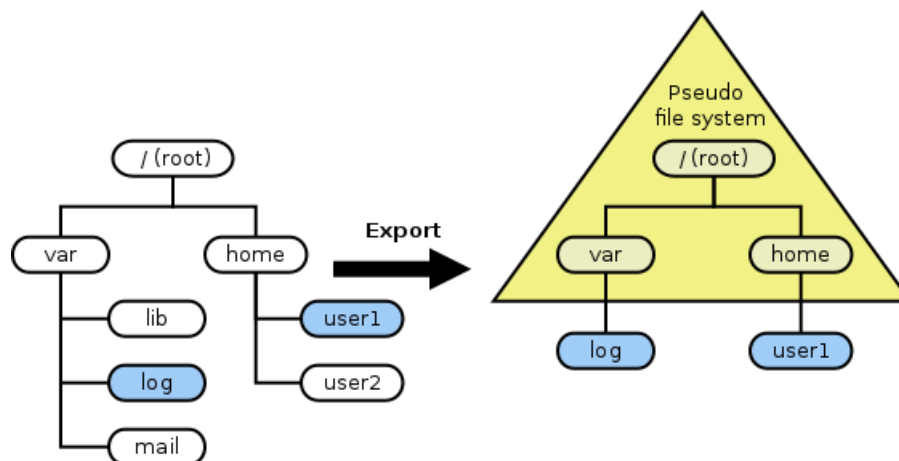
Obrázek 8: Theo de Rath k NFSv4 (Raadt, 2010)

Zatímco NFSv3 pro svou činnost spoléhá na externí protokoly a programy (mountd, portmap, ...), NFSv4 v sobě integruje všechny potřebné funkce včetně zámků souborů. Tato integrace přispívá ke zjednodušení zabezpečení NFS serverů (Haynes, 2015, s. 1).

2.2.3 Nové vlastnosti NFSv4

Pseudo-file system

Při exportování (sdílení) více než jednoho lokálního umístění musel NFS klient připojit každé z těchto umístění zvlášť. S NFSv4 tato nutnost odpadá. Nově je nad jednotlivými umístěními vytvářen tzv. „pseudo-file system”, který slouží jako přemostění mezi exportovanými umístěními. Klientovi je tak umožněno tato vzdálená umístění libovolně procházet, přičemž je každý klient samozřejmě limitován svými oprávněními. Identifikace klientů již neprobíhá na základě UID, server namísto toho provádí mapování UID na uživatelská jména (Haynes, 2015, s. 78-79).



Obrázek 9: Pseudo-file system

V rámci pseudo-souborového systému má uživatel oprávnění pouze pro čtení. Při procházení hierarchické struktury je omezen výhradně na adresáře vedoucí k exportovanému umístění. Především v linuxové implementaci se stalo dobrým zvykem minimalizovat velikost pseudo-souborového systému pouze na jediné umístění `/exports`, do nějž jsou symbolickými odkazy „připojena“ lokální umístění určená k exportu (Haynes, 2015, s. 78).

Transportní protokol

Network File System historicky stavěl na protokolu UDP. Ačkoli v NFSv3 přibyla podpora TCP, nebylo jeho využití povinné. UDP bylo přijímáno jako jednodušší a rychlejší řešení pro transportní vrstvu, oproti TCP ale trpělo neexistencí mechanismu řešícího zahlcení sítě (Callaghan, 1995, s. 100).

Aplikace pracující nad UDP také musí samostatně řešit problémy chybějících či porušených datagramů. Protokol nijak neřeší bezpečné doručení nebo pořadí datagramů. S použitím UDP také souvisí problém nazývaný „silent data corruption“ (KERRISK, 2012).

Tento problém spočívá ve fragmentaci UDP datagramů do paketů na síťové vrstvě. Pakety tvořící jeden datagram jsou označeny identifikátorem o délce 16 bitů. Přijímající strana před sestavením datagramu čeká až 30 sekund na všechny pakety. Při odeslání více než 65536 paketů během 30 sekund může dojít k odeslání paketu se správným identifikátorem, ale neodpovídajícím datagramem. Tento problém souvisí především s použitím UDP pro transport přes vysokorychlostní spojení (Gigabit LAN) (KERRISK, 2012).

NFSv4 pro transportní vrstvu vyžaduje protokol standardizovaný organizací IETF, přičemž tento protokol musí obsahovat mechanismy řešící přetížení síťového spojení. Těmito protokoly jsou Stream Control Transmission Protocol (SCTP) a dobře známý TCP. V rámci kompatibility musí každá implementace NFSv4 podporovat TCP (Haynes, 2015, s. 25).

Změn doznal i systém řešící výpadky komunikace mezi serverem a jeho klienty. Předšlé verze spoléhaly na časovače, po jejichž vypršení došlo k opakovanému volání procedury a tedy přeposlání požadavku. Požadavky byly jednoduše přeposílány do té doby, dokud server nezaslal odpověď. Protože NFSv4 užívá spolehlivý transport dat, nemusí se klient spoléhat na opakování požadavků jako na jediný mechanismus „detekce a řešení“ výpadku. Pokud klient neobdrží informaci o přerušení spojení se serverem, pak se nesmí uchýlit k opakovanému volání.

Namísto toho klient volá tzv. „null procedure“. Tímto voláním klient serveru oznamuje, že došlo k vypršení časového limitu pro volání vzdálené procedury. Klient se tím serveru jednoduše „připomíná“. Pokud server na volání „nulové procedury“ nereaguje, pravděpodobně tak došlo k jeho výpadku. Přerušení transportního spojení je poté klientem detekováno, na což reaguje navázáním nového. Až v rámci nového spojení může klient opakovaně volat původní proceduru. Pokud ale klient na volání nulové procedury obdrží odpověď, nezbyvá mu, než déle čekat na „opožděnou“ odpověď od serveru (Haynes, 2015, s. 26).

Odklon od mapování portů

Použití NFS napříč různými sítěmi typicky znamenalo významné změny v nastavení firewallu, který odděloval klienty od serveru a naopak. Nejenom, že musel být klientům přístupný program „portmap“ na portu 111, musel být otevřen i port pro samotný NFS server (typicky 2049) a další RPC programy zajišťující zamykání souborů (lockd) a připojování souborových systémů (mountd).

Integrace všech těchto modulů do monolitického serveru umožňuje soustředit veškerou komunikaci od klientů směrem k serveru na port 2049. Díky tomuto je snadné přizpůsobit nastavení firewallu i případné přesměrování portů na zařízeních provádějících překlad adres (Haynes, 2015, s. 25).

Přechod na UTF-8

Nejen pro názvy souborů a adresářů užívá NFSv4 kódování UTF-8, které je zpětně kompatibilní se sedmi bitovým ASCII kódováním. Odklon od ASCII byl vynucen podporou delších abeced, se kterými se v původním protokolu nepočítalo (Haynes, 2015, s. 10).

Složená RPC volání

NFSv4 přineslo novinku v podobě hromadného transportu hned několika RPC volání. Samostatná transakce pro zpracování každého vzdáleného volání způsobuje neúměrný nárůst latence především při komunikaci přes internet. Sloučení několika volání do jedné zprávy tak umožňuje zvýšit efektivitu serveru při nižším zatížení sítě (Haynes, 2015, s. 10).

Delegování

Velkou novinkou NFSv4 je delegování „odpovědnosti“ za zpracování souborů. Operace nad delegovaným souborem jsou klientem vykonávány lokálně v jeho mezipaměti, čímž je serveru umožněno vykonávat jiné operace a tedy dále zvyšovat jeho výkon. Delegování klientovi zajišťuje výlučný přístup k souboru, přičemž operace čtení a zápisu mohou být vykonávány bez další interakce se serverem. Server může delegování souboru kdykoli zvrátit asynchronním voláním klientské stanici (Haynes, 2015, s. 13).

Bezpečnost

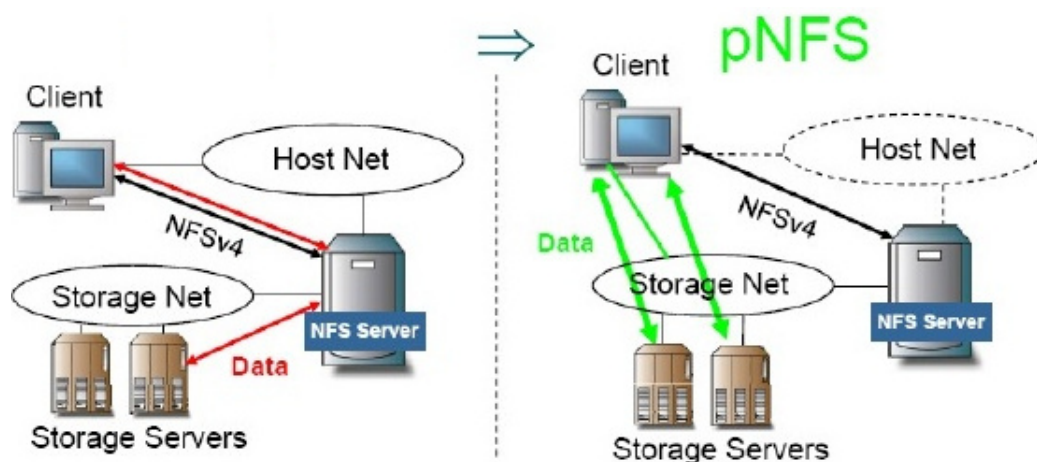
Jedním z důvodů chladného přijetí NFSv4 byl velký důraz na bezpečnost přenosu dat. Standard totiž jasně definuje použití bezpečnostního protokolu RPCSEC_GSS jako povinné. Ačkoli ho tedy každá implementace NFSv4 musí obsahovat, není jeho použití povinné. Jeho nasazení je pouze výrazně doporučeno v sítích WAN. Ostatní bezpečnostní mechanismy dobře známé z předešlých verzí NFS (AUTH_NONE, AUTH_SYS, AUTH_DH, ...) „mohou“ být dle standardu obsaženy také. Ačkoli je tedy přítomnost silného bezpečnostního mechanismu povinná, jeho použití závisí čistě na implementaci a posléze na správci serveru (Haynes, 2015, s. 25-29).

2.2.4 Parallel NFS

Paralelní NFS neboli pNFS bylo jednou z velkých novinek dílčí revize NFSv4.1 z ledna roku 2010. Důvodem vzniku pNFS byly potřeby velkých datových center. I přes značné rozšíření přestal výkon NFSv3 dostávat nárokům data center a začaly se objevovat všemožné modifikace s cílem zlepšit škálovatelnost stávajících implementací (SHEPLER, 2010, s. 1).

Cílem pNFS je oddělit informace o datech od samotných dat. Zatímco tzv. „metadata server“ disponuje informacemi o umístění a organizaci souborů, samotná data jsou umístěna separátně v datovém úložišti spravovaném příslušným serverem (SHEPLER, 2010, s. 307).

Tento server poté poskytuje informace potřebné k přístupu k souborům v podobě tzv. schématu (layout). Klient s metadata serverem komunikuje pomocí speciálního „control“ protokolu a získává schéma popisující uložení dat. Standard NFSv4.1 popisuje tři schémata (SHEPLER, 2010, s. 309-334).



Obrázek 10: Parallel NFS (ZHANG,2009, s. 7)

- Blokové schéma (Block Layout) do nějž spadá například iSCSI,
- objektové schéma (Object Storage Device),
- souborové schéma (totožné s obyčejným NFSv4).

K samotným souborům poté klient přistupuje přímo na základě použitého schématu, který určuje příslušný „storage-access” protokol. Transfer dat již ale probíhá bez dalšího zapojení metadata serveru. K tomu dochází pouze v případech, kdy klient nedisponuje podporou pNFS, čímž je zajištěna i zpětná kompatibilita. Metadata server takovým klientům poskytuje všechny standardní služby bez toho, aby si byl klient vědom všech možností a architektury úložiště (SHEPLER, 2010, s. 308).

3 Návrh vlastního protokolu

Protokol, ať už síťový nebo jakýkoli jiný, je definován jako množina zpráv s pevně definovaným formátem a akcemi vykonávanými dvěma nebo více mezi sebou komunikujícími entitami v reakci na přijetí těchto zpráv. Alespoň takto termín „protokol” popisují James F. Kurose a Keith W. Ross ve své knize „Computer Networking - A Top-Down Approach Featuring the Internet”.

„A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message.”

Gerard J. Holzmann popisuje protokol jako „dohodu o výměně informací v distribuovaném systému”, přičemž jeho definici přirovnává k definici jazyka. Protokol

- definuje přesný formát validních zpráv (syntaxi),
- dále procedurální pravidla pro výměnu dat (gramatiku)
- a slovník validních zpráv s jejich přesným významem (sémantiku).

Ve své knize „Design and validation of computer protocols” Holzmann zmiňuje 5 základních elementů, které by měly být obsaženy ve specifikaci síťového protokolu.

1. Služba poskytovaná protokolem. Proč protokol vznikl?
2. Předpoklady o prostředí. Kde a v jaké podobě by měl být protokol nasazen?
3. Slovník zpráv používaných protokolem.
4. Formát těchto zpráv. Například použité kódování.
5. Pravidla pro vzájemnou výměnu zpráv a následné akce.

Dánská „Aarhus University” ve svém kurzu „Protocol design” popisuje pět pokynů pro návrh protokolů, které volně korespondují s některými z Holzmannových elementů. Tyto pokyny lze podat v podobě otázek, na které je nutno si na počátku vývoje odpovědět. Výsledný protokol bude formován odpovědí na každou z nich a od původní představy autora se může značně lišit.

- Jaká má být pozice nového protokolu ve stávajícím balíku protokolů?
 - Od odpovědi na tuto otázku se očekává jasná představa o tom, na jaké vrstvě síťové architektury má protokol pracovat a s jakými prvky sítě bude komunikovat. S tím přímo souvisí formát zpráv a jejich zapouzdření do datových jednotek (PDU) na ostatních vrstvách datového modelu.
- Jaké budou vnitřní stavy protokolu a jak to ovlivní jeho chování?
 - Každý druh komunikace je založen na reakcích na nastalou událost či stav. Proto je nutné identifikovat, jakých stavů bude aplikace implementující daný protokol nabývat a jak bude reagovat na jeho změny v důsledku vnitřních nebo vnějších událostí.
- Alokace paměti a spolupráce s operačním systémem.
 - Alokování paměti bude probíhat staticky nebo za běhu?
 - Bude protokol užívat systémovou autentizaci, IPC nebo časovače?
- Modularita programu?
 - Vývoj programu implementujícího nový protokol si může vyžádat použití vláken nebo procesů pro rozdělení odpovědností. Vývojář si musí být vědom toho, zda a jak tím může být ovlivněna podoba protokolu.

3.1 Easy Sharing Protocol

Název protokolu vznikl na základě jeho účelu, tedy tak, jak už je ve světě počítačových sítí zvykem. Myšlenkou stojící za ESP bylo od počátku pro uživatele jednoduché zasílání souborů na stanice v lokální síti. Pojmenování protokolu vzniklo jako první hned po úvodní myšlence. Ačkoli existuje nespočet protokolů, ať už síťových nebo jiných (kniha „Network Protocol Handbook” jich popisuje na 300), název i zkratka ESP se jeví jako unikátní. Nebylo proto třeba se nad identifikátorem dále zamýšlet.

Pouze dvě široce známé technologie užívají stejnou zkratku. První je elektronický stabilizační program, který pomáhá předcházet smykům u automobilů. Tou druhou je „Encapsulating Security Payload” z balíku protokolů IPsec.

IPsec neboli „Internet Protocol security” je technologie zajišťující autentizaci a šifrování IP paketů na třetí vrstvě OSI modelu. Pokud se budeme držet specifikace Gerarda Holzmanna, bude mít ESP protokol následující elementy.

3.1.1 Služba poskytovaná protokolem

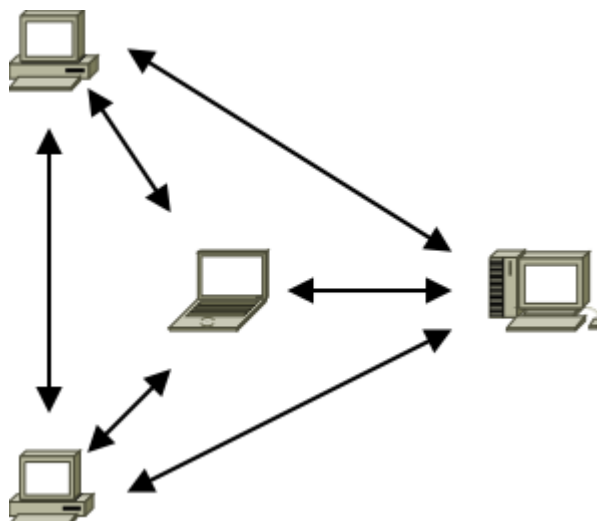
Protokol bude sloužit k přenosu libovolných souborů v lokální síti s možností ověřit pravost souboru. Cílem protokolu je poskytnout jednoduchý nástroj pro uživatelsky nenáročné „point-to-multipoint” sdílení souborů. Implementace nesmí předpokládat povahu ani typ přenášených dat, ke všem musí přistupovat stejným způsobem. Přenos každého souboru musí být před jeho odesláním explicitně vyžádán protistranou.

3.1.2 Předpoklady o prostředí

Cílovým prostředím pro nasazení ESP jsou skupiny dvou a více uživatelů náhodně sdílejících rozličné druhy souborů bez možnosti přístupu k dedikovanému síťovému úložišti. Dalším předpokladem je nehomogenní prostředí na úrovni operačního systému. Ačkoli implementace ESP mohou být na platformě závislé, měly by být schopné mezi sebou komunikovat bez ohledu na operační systém pohánějící každou z nich. Při návrhu protokolu proto nebylo přihlíženo k vlastnostem jedné vybrané platformy.

Protokol je závislý pouze na balíku protokolů TCP/IP. Pro adresování zpráv na třetí vrstvě OSI modelu ESP spoléhá výhradně na protokoly IPv4 a v budoucnu případně na IPv6. Na čtvrté a tedy transportní vrstvě protokol využívá jak datagramové služby UDP, tak protokolu spolehlivého přenosu dat TCP. Důvodem závislosti na TCP je jeho schopnost řízení datového toku v závislosti na vytížení sítě.

Z pohledu hierarchie protokolů předpokládá rovnoprávné postavení všech uživatelů. V síti implementující ESP není žádný z uzlů považován za centrální prvek. Sdílení souborů řídí každý uzel nezávisle na ostatních a je plně v kompetenci uživatelů jednotlivých stanic. Nejedná se tedy o architekturu „klient-server” a implementace by od uživatele neměla očekávat žádné pokročilé znalosti práce se sítí. Výsledný program by měl být pro uživatele intuitivní a před svým spuštěním by neměl vyžadovat ruční konfiguraci.



Obrázek 11: Peer-to-peer architektura

Cílem protokolu je nabídnout společnou řeč pro stanice poháněné nejznámějšími operačními systémy ze segmentu osobních počítačů. Předpokládá se podpora přinejmenším systémů

- Windows (ve svých aktuálně podporovaných edicích)
- a linuxových distribucí pro osobní počítače.

3.1.3 Slovník zpráv

Protokol ESP obsahuje 10 zpráv s přesně definovaným významem. Protokol zahrnuje i zprávy, které sloužily k testovacím účelům během vývoje programu. Ačkoli nebyly z protokolu odstraněny, pro jeho funkci nejsou nijak důležité.

3.1.4 Formát zpráv

Jednotlivé zprávy jsou definované jako hodnoty výčtového datového typu. Hodnota je, jak je u enumerátorů zvykem, určena pořadím zpráv. Proto je důležité, aby jednotlivé implementace užívaly společný zdrojový soubor definující příslušný výčtový typ.

Po síti jsou zprávy přenášeny ve formě řetězce (string) odpovídajícího standardu UTF-8. Při použití nástroje pro analýzu síťového provozu Wireshark by například zpráva „HELLO” měla nabývat hexadecimální hodnoty 30.

<i>Zpráva</i>	<i>Význam</i>
HELLO	Aplikace touto zprávou oznamuje svou přítomnost v síti. Zpráva je odesílána automaticky bez vědomí uživatele.
ITEMS	Předchází seznamu souborů sdílených vzdáleným uzlem. Upozorňuje program k přijetí souvislého proudu dat.
PING	Ověřuje funkci navázaného síťového spojení. Tato zpráva slouží výhradně k testovacím účelům.
NAME	Vyžaduje název souboru, který si protistrana přeje stáhnout. V reakci na přijetí je zaslán řetězec s názvem souboru.
_FILE	Informuje vzdálený uzel o následující akci, tedy odeslání souboru. Přijímající strana se musí připravit na přijetí souboru.
CRC	Slouží k vyžádání kontrolního součtu přijatého souboru. Po jejím přijetí následuje název souboru jehož součet je vyžadován.
SUM	Informuje protistranu o dokončení požadovaného kontrolního součtu. Po jejím odeslání je odeslán i příslušný kontrolní součet.
BYE	Instance programu informuje protistranu o svém ukončení. Po přijetí této zprávy je záznam o odesílateli odstraněn z programu.
ACK	Slouží k informování protistrany o přijetí jiné zprávy.

Tabulka 18: Seznam definovaných zpráv

Algoritmus 3 ESP zprávy

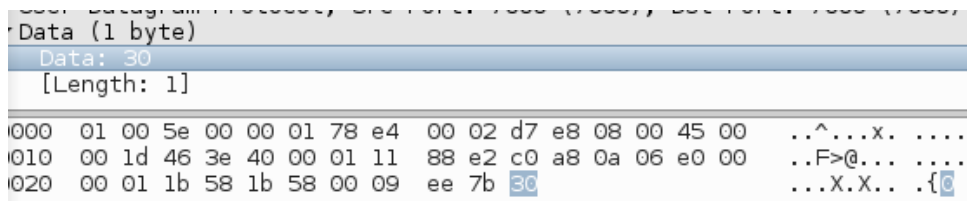
```
#ifndef MESSAGES_H
#define MESSAGES_H
enum MSG {HELLO,ITEMS,PING,NAME,_FILE,CRC,SUM,BYE,ACK};
#endif // MESSAGES_H
```

3.1.5 Pravidla pro výměnu zpráv

HELLO

Jednotlivé uzly zahajují komunikaci pomocí HELLO zpráv. Ty plní funkci tzv. objevování sousedů neboli „neighbour discovery”, což je koncept dobře známý například ze směrovacích protokolů OSPF a proprietárního EIGRP. Účelem těchto zpráv není navázat funkční spojení, pouze oznámit přítomnost uzlu v síti.

Každý uzel odesílá HELLO zprávy v podobě UDP datagramů na vzdálený port 7000. Cílovou adresou jsou všechny stanice v lokální síti. Pro IPv4 je to tzv. „all-hosts multicast” představovaný adresou 224.0.0.1. Obdobou v novém protokolu IPv6 je „all-nodes multicast” s adresou FF01::1. Pro adresování všech zařízení v síti by se nabízelo použití všesměrové adresy (broadcast). Její tvar se ale na třetí vrstvě OSI modelu odvíjí od použitého adresního rozsahu.



Obrázek 12: Zachycená zpráva HELLO

Jejím použitím by navíc HELLO zprávy odcházely na všechna zařízení v síti, včetně směrovačů, administrativních rozhraní přepínačů a jiných zařízení, přičemž toto chování není žádoucí. Dalším „problémem“ je absence všesměrových adres v IPv6.

Všesměrové adresy byly nicméně z protokolu IPv6 odstraněny záměrně. Jejich funkce byla překonána konceptem skupinových (multicast) adres, kterým se v IPv6 dostalo vřelejšího přijetí než v IPv4. Zatímco v době vzniku a prvotního šíření IPv4 se o použití skupinových adres moc nepřemýšlelo, u IPv6 je jejich podpora povinná. Důvodem odchýlení se od všesměrového vysílání směrem ke skupinovému jsou právě rozdílné druhy zařízení v síti. Zařízení nespádající do cílové skupiny tak nebudou zatěžovány zbytečným provozem, čímž jsou šetřeny jejich systémové zdroje.

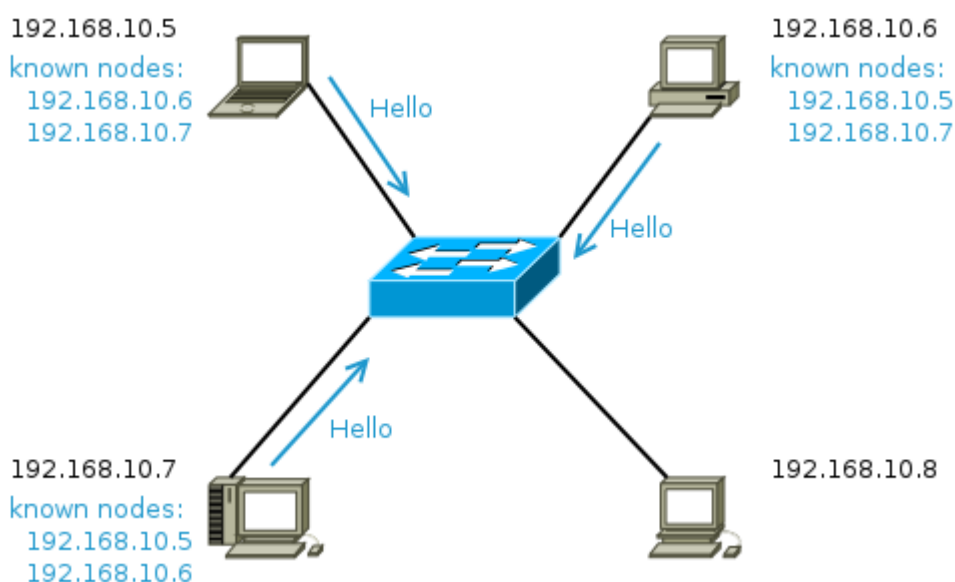
Tabulka 19: Hodnoty zpráv a UTF-8

<i>Zpráva</i>	<i>Znak</i>	<i>UTF-8 (hex.)</i>	<i>Význam</i>
HELLO	0	0x30	číslice nula
ITEMS	1	0x31	číslice jedna
PING	2	0x32	číslice dva
NAME	3	0x33	číslice tři
_FILE	4	0x34	číslice čtyři
CRC	5	0x35	číslice pět
SUM	6	0x36	číslice šest
BYE	7	0x37	číslice sedm
ACK	8	0x38	číslice osm

Protokol OSPF užívá HELLO zprávy primárně k navazování tzv. „sousedství“ (adjacency) s ostatními směrovači. Každá instance OSPF si uchovává tabulku záznamů popisujících směrovače nacházející se v jeho okolí. Na základě obdržených HELLO zpráv poté modifikuje příslušný záznam. Po přijetí zprávy od směrovače, jehož RouterID není v tabulce obsaženo, zahajuje směrovač proces formování sousedství. Ten zahrnuje několik dalších výměn HELLO zpráv a průchod záznamu o sousedním směrovači celou řadou stavů (Moy, 1994, s. 45-46).

Proces formování „dobrých sousedských vztahů“ nicméně u OSPF nekončí u identifikace okolních směrovačů. Status každého směrovače v tabulce sousedů ovlivňuje jakousi úroveň spolupráce mezi jednotlivými směrovači. Při dosažení plného sousedství (Full adjacency state) disponují všechny instance OSPF totožnou představou o podobě sítě, na základě které jsou poté vykonávány operace směrování (Moy, 1994, s. 75).

V případě ESP je způsob použití HELLO zpráv o něco jednodušší. Po přijetí prvního UDP datagramu přidává přijímající strana záznam do svého seznamu známých uzlů (known nodes). Jednotlivé uzly mezi sebou nevytváří komplexní sousedské vztahy, pro další funkci je nicméně nutné, aby obě strany ve svém seznamu známých uzlů disponovaly záznamem o protějšku. To vše se musí dít bez jakéhokoli trvalého spojení, tedy bez použití TCP komunikace.

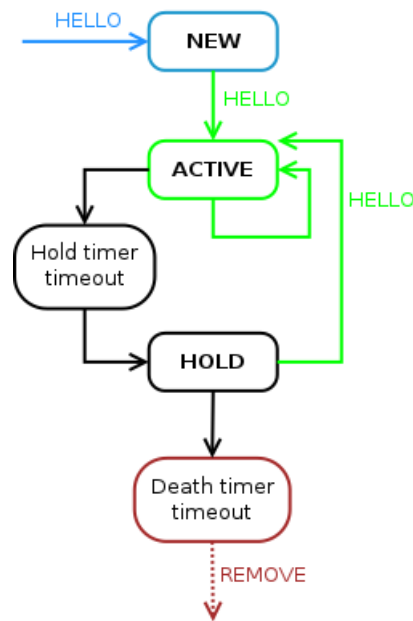


Obrázek 13: HELLO zprávy

K navázání TCP spojení dochází až v momentě příjmu nebo odeslání seznamu sdílených souborů. Tato akce by měla být iniciována uživatelem aplikace, v závislosti na implementaci ji ale lze i automatizovat. Pro potřeby testování dostupnosti lze TCP spojení navázat i odesláním zprávy Ping. To je ale možné pouze v případě přítomnosti záznamu o druhé straně v seznamu známých uzlů.

Obsah seznamu známých uzlů se řídí výhradně pravidelným příjmem HELLO zpráv od ostatních instancí ESP běžících na okolních stanicích v síti. Implementace ESP musí odeslat HELLO zprávu každých pět vteřin, přičemž po jejím přijetí dochází k jedné ze dvou situací.

Přijímající strana nejdříve na základě adresy odesílatele zkontroluje prvky svého seznamu známých uzlů. Pokud je pro požadovanou adresu nalezen záznam, dojde k resetování jeho časovače přidržení a jeho stav je (znovu) nastaven na „aktivní“. Pokud pro danou adresu záznam neexistuje, je vytvořen nový. Pro nový záznam je spuštěn časovač přidržení a jeho status je nastaven na „nový“. Z „nového“ na „aktivní“ se status záznamu mění přijetím v pořadí druhé HELLO zprávy ze stejné adresy.



Obrázek 14: Status v závislosti na časovačích

- Časovač *HELLO* zpráv slouží k pravidelnému odesílání zpráv oznamujících přítomnost uzlu v síti. Interval jejich odeslání je stanoven na 5 vteřin a odpovědnost za jejich odeslání by měl nést modul „objevování sousedů“.
- Časovač *přidržení* (hold timer) určuje dobu, po jakou bude záznam o vzdáleném uzlu držen v paměti. Jeho hodnota začíná na patnácti sekundách a je resetována při každém přijetí *HELLO* zprávy z příslušné adresy. Každý záznam odpovídající jednomu z okolních uzlů tedy disponuje vlastním časovačem přidržení. Díky jeho prodlevě, která odpovídá trojnásobku časovače *HELLO* zpráv, umožňuje překonat krátkodobé výpadky síťového spojení. V případě vypršení tohoto časovače dochází ke změně statutu záznamu o vzdáleném uzlu na „HOLD“, čímž je indikováno jeho podržení v paměti, ačkoli již není aktivní. Současně dochází k aktivaci časovače smrti.

- Časovač smrti je spuštěn pro každý uzel samostatně a to v důsledku vypršení jeho časovače přidržení. Časovač začíná na patnácti vteřinách a při jeho vypršení dochází k odstranění záznamu o již nekomunikujícím uzlu. Záznam není odstraněn pouze v tom případě, kdy se vzdálený uzel opět ohlásí zasláním HELLO zprávy. Status záznamu se opět změní na aktivní a se vzdáleným uzlem je možné znovu komunikovat.

ITEMS

Seznamu sdílených souborů předchází zpráva ITEMS, přičemž se její příjemce musí připravit na okamžité přijetí textového řetězce proměnné délky v kódování UTF-8. Jejímu odeslání typicky předchází navázání TCP spojení, jehož iniciátorem je odesílatel této zprávy.

Seznam sdílených souborů je odesílán v podobě textového řetězce proměnné délky s pevně specifikovaným formátem. Seznam pro každý v něm uvedený soubor nese jak název a koncovku souboru, tak informaci o jeho velikosti. Tuto informaci je po přijetí řetězce nutné převést opět na číselnou hodnotu a korektně přiřadit k příslušnému souboru, respektive k záznamu popisujícímu daný soubor.

Tvorba souvislého řetězce představujícího seznam souborů je definován jako krátký proces o dvou krocích. Prvním krokem je vytvoření tzv. popisů či „popisovačů“ jednotlivých souborů. Popisovače jsou tvořeny názvem příslušného souboru s koncovkou a textovou reprezentací jeho velikosti v bajtech. V rámci každého popisovače jsou název a velikost odděleny unikátní sekvencí znaků „:/“, známou jako FDM (File Description Message). Tato sekvence ulehčuje pozdější oddělení v popisovači obsažených informací přijímající stranou.

Tabulka 20: Příklady popisovače souboru

<i>Název souboru</i>	<i>Velikost v bajtech</i>	<i>Popisovač</i>
archiv.zip	13 481 372	„archiv.zip:/13481372”
foto.png	583 229	„fotografie.png:/583229”
video.avi	788 520 293	„video.avi:/788520293”

Obdobný účel, jako má sekvence FDM, tedy oddělení dvou nesourodých informací o souboru, plní i sekvence IDM (Item Description Message). Ta v souvislém řetězci odděluje

jednotlivé popisovače tvořící prvky seznamu sdílených souborů. Tato sekvence je tvořena znaky „/;“.

Tabulka 21: Vytvoření řetězce prvků

<i>Prvky seznamu</i>
„archiv.zip/;/13481372“
„foto.png/;/583229“
„video.avi/;/788520293“
<i>Kompletní řetězec</i>
„archiv.zip/;/13481372;/fotografie.png/;/583229;/video.avi/;/788520293“

PING

K testování funkčnosti TCP spojení sloužila během vývoje zpráva PING. Zpráva nijak nezávisí na zprávách ostatních a ani nijak nezasahuje do jejich funkcí. Ačkoli je pro funkci kompletního protokolu nadbytečná, lze tuto zprávu díky její nezávislosti zahrnout do finální implementace. Pokud není do jejího odeslání spojení navázáno, musí být ustaveno před jejím prvním odesláním. Zpráva PING je jinak považována za rezervovaný příkaz.

NAME

Zprávu NAME lze odeslat pouze tomu uzlu, který s původcem této zprávy sdílí alespoň jeden soubor. Aby mohl uzel odeslat zprávu NAME, musí od svého protějšku alespoň jednou obdržet zprávu ITEMS následovanou seznamem sdílených souborů. Po odeslání této zprávy musí následovat název požadovaného souboru, který má být od protějšku stažen. Příjemce této zprávy se musí připravit na přijetí názvu souboru, po jehož úspěšném přijetí je zahájen proces odeslání souboru.

_FILE

Odesláním této zprávy uzel svému vzdálenému protějšku oznamuje odeslání souboru. Přijímající strana se musí připravit na okamžité přijetí souboru o předem známé velikosti. Soubory jsou odesílány v surové podobě a do socketu jsou zapisována tak, jak jsou čtena z pevného úložiště. Protokol nespecifikuje jakým způsobem mají být soubory z úložiště čteny, ani jak na něj mají být ukládány. Z důvodu efektivního využití systémových zdrojů je nicméně doporučeno data číst a zapisovat po blocích pevné velikosti.

CRC

Zprávou CRC aplikace žádá svůj protějšek o výpočet kontrolního součtu souboru, který si od něj dříve stáhla. Vyžádání kontrolního součtu nicméně není povinné a nemělo by nutně navazovat na stažení konkrétního souboru. Požadavek na ověření souboru by měl být odeslán pouze v reakci na přání uživatele a to nezávisle na pořadí stažených souborů. Právě proto je požadavek o kontrolní součet následován názvem souboru, pro nějž má být výpočet proveden.

Výpočet může probíhat na obou uzlech simultánně, případně může přijímající strana provést svůj výpočet až po přijetí kontrolního součtu od odesílatele souboru. Protokol nicméně poradí, v jakém mají být jednotlivé výpočty provedeny, nijak nespecifikuje. Toto chování je tedy závislé na konkrétní implementaci a přáních jejího vývojáře.

Pro ověřování pravosti souborů byly zvažovány algoritmy SHA-256 a SHA-512 z rodiny hašovacích funkcí „Secure Hash Algorithm 2“. Ačkoli je pro ověřování pravosti souborů stále široce užívána dobře známá hašovací funkce MD5, bylo od jejího použití vzhledem k velkému množství známých kolizí upuštěno.

Tabulka 22: Kontrolní součty na systému Debian GNU/Linux

<i>Soubor</i>	<i>Velikost</i>	<i>Funkce</i>	<i>Doba výpočtu</i>
jooq.zip	13,5 MB	MD5	44 ms
		SHA-256	1734 ms
		SHA-512	3861 ms
debian-9.0.0-amd64-xfce-CD-1.iso	678,4 MB	MD5	124 ms
		SHA-256	5796 ms
		SHA-512	12889 ms
ubuntu-17.04-desktop-amd64.iso	1517,8 MB	MD5	88 ms
		SHA-256	3874 ms
		SHA-512	8728 ms

Před finálním výběrem algoritmu pro kontrolu korektního přenosu souboru byla provedena krátká řada testů porovnávajících výkon jednotlivých hašovacích funkcí. Testy byly prováděny na operačním systému Debian GNU/Linux 8.8 s architekturou amd64 a to s použitím interních funkcí systému a multiplatformních vývojových knihoven Qt 5.3, které byly později zvoleny pro vývoj ukázkového programu.

Pro větší názornost výsledků byly pro testování zvoleny soubory o velikosti od několika málo megabajtů po jeden a půl gigabajtu. Jak je zřejmé z jednotlivých výsledků, vývojové

knihovny nedosahují stejného výkonu jako nativní funkce, prezentují ale zajímavý „paradox“. Při použití algoritmu SHA-512 dosahují lepších výsledků, než při použití výkonově „méně náročného“ algoritmu SHA-256.

Jedná se nicméně o dobře známý fakt. Díky 64-bitovému bloku dat v každé iteraci může algoritmus SHA-512 na 64-bitových platformách dosahovat až 1,5 násobku výkonu algoritmu SHA-256. Ten vstupní data čte pouze po blocích o délce 32 bitů, v důsledku čehož musí provádět podstatně více iterací, i když méně časově náročných.

Ačkoli jsou 32-bitové systémy pomalu na ústupu, je nutno s nimi počítat, neboť jsou v průmyslu stále značně zastoupeny. Právě proto bylo rozhodnuto o použití hašovacího algoritmu SHA-256.

Tabulka 23: Kontrolní součty pomocí knihoven Qt

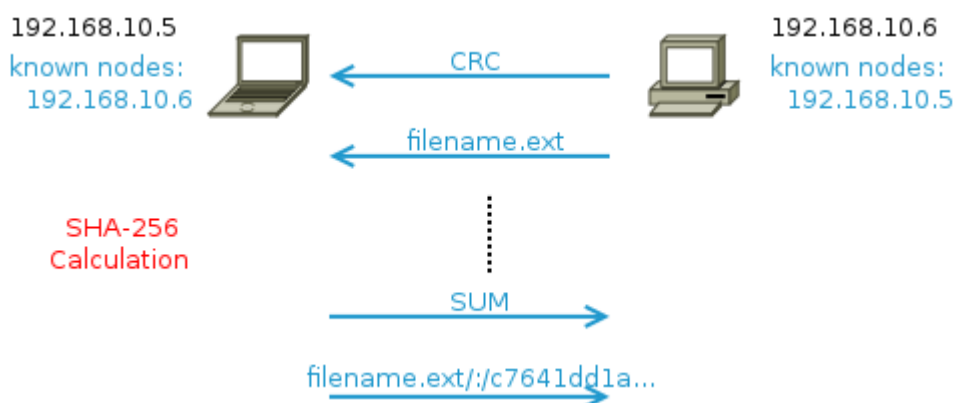
<i>Soubor</i>	<i>Velikost</i>	<i>Funkce</i>	<i>Doba výpočtu</i>
jooq.zip	13,5 MB	MD5	41 ms
		SHA-256	200 ms
		SHA-512	139 ms
debian-9.0.0-amd64-xfce-CD-1.iso	678,4 MB	MD5	1896 ms
		SHA-256	9320 ms
		SHA-512	6734 ms
ubuntu-17.04-desktop-amd64.iso	1517,8 MB	MD5	4182 ms
		SHA-256	20606 ms
		SHA-512	15252 ms

SUM

Dokončení výpočtu kontrolního součtu protistrana oznamuje zasláním zprávy SUM. Po obdržení této zprávy se přijímající strana musí připravit k okamžitému přijetí popisovače s kontrolním součtem. Ten je opět zasílán v podobě textového řetězce v kódování UTF-8 ve formátu „filename.ext/:c7641dd1a...“.

Protokol nijak nespecifikuje, jak má přijímající strana s kontrolním součtem naložit, nicméně se očekává okamžité porovnání s lokálním kontrolním součtem. Po porovnání obou součtů se předpokládá informování uživatele o výsledku tohoto srovnání a tedy o tom, zda byl přijatý soubor zapsán na specifikované úložiště korektně.

Je zcela na uživateli, jak se zachová v situaci, kdy jednotlivé kontrolní součty nesouhlasí. Objevení neidentické kopie nemá žádné přímé dopady na vztah ani spojení dvou komunikujících uzlů. Je-li spojení mezi uzly stále aktivní, pak se uživatel může pokusit o opakované stažení porušeného souboru.



Obrázek 15: CRC a SUM zprávy

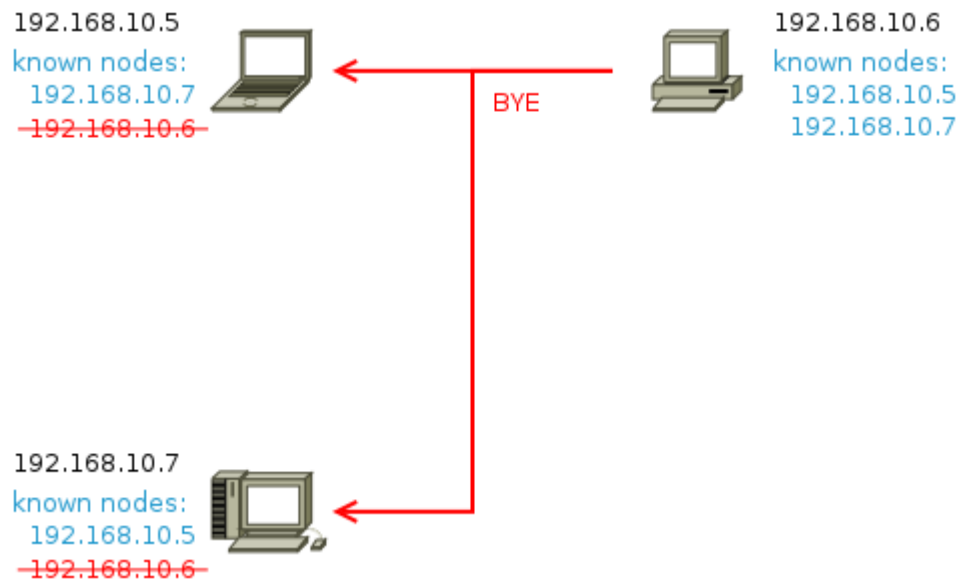
BYE

Touto zprávou se jednotlivé uzly „loučí“ se svými protějšky v síti. Zprávy jsou odesílány při zastavení serveru a jejich účelem je obejít funkci časovačů. K jejich odeslání dochází ve chvíli, kdy je jeden ze vzájemně komunikujících uzlů ukončen akcí uživatele. Nejedná se tedy o jakýsi mechanismus „obnovení po selhání“. Dojde-li k selhání uzlu, pak je jeho nedostupnost v síti indikována absencí HELLO zpráv odesílaných z jeho síťové adresy. Pokud dojde k vypršení všech časových intervalů na sousedních uzlech, jsou odstraněny všechny záznamy o existenci příslušného uzlu.

Spoléhat se na časovače při ukončení aplikace je nežádoucí. Udržování a aktualizace „neplatného“ záznamu o protějším uzlu konzumuje systémové zdroje každé stanice v síti. Pokud to tedy situace dovoluje, je vhodné svou budoucí nepřítomnost oznámit všem komunikujícím prvkům s předstihem.

Ke korektnímu ukončení dochází výhradně v reakci na požadavek uživatele, po jehož vzněšení je zpráva BYE odeslána všem dostupným uzlům. Aplikace před svým ukončením prochází seznam dostupných uzlů s cílem jejich vymazání a uvolnění zdrojů. Před vymazáním každého záznamu je příslušnému protějšku odeslána zpráva BYE, načež dochází k uzavření spojení a vymazání síťového soketu.

Na přijímající straně vyvolá přijetí zprávy BYE požadavek na vymazání „sama sebe”. Příjemcem zprávy je totiž záznam o vzdáleném uzlu. Ten po vyhodnocení této zprávy vznesne požadavek ke svému vymazání. Správce záznamů o uzlech v reakci na tento požadavek vyzve daný záznam k uvolnění všech otevřených zdrojů a posléze provede jeho vymazání. Uživatel je o této akci okamžitě informován odebráním vizuální reprezentace záznamu v uživatelském prostředí.



Obrázek 16: BYE zprávy

ACK

Zpráva ACK byla používána k testovacím účelům během vývoje protokolu a implementace demonstrační aplikace. Její zaslání sloužilo k potvrzení úspěšného přijetí jiné zprávy. Ačkoli mohla být zpráva z protokolu odstraněna, namísto toho byla ve specifikaci ponechána v podobě rezervované zprávy.

4 Standardizace protokolu

Každý standard začíná svou cestu v podobě technické specifikace. Specifikace každého budoucího protokolu jsou dostupné na úložištích organizace IETF (Internet Engineering Task Force) v podobě tzv. „internetových konceptů“ neboli „draftů“ (Bradner, 1996, s. 8). Koncept je ranou vývojovou fází každé specifikace a s jako takovými musí být s těmito dokumenty zacházeno.

Koncepty mohou být v úložišti kdykoli nahrazeny svou novější verzí, popřípadě z něj mohou být po příslušném rozhodnutí odstraněny. Z těchto důvodů se na koncepty nesmějí odkazovat žádné publikovatelné práce ani dokumentace produktů výrobců síťových zařízení (Bradner, 1996, s. 8). Pokud není koncept nahrazen svou novější verzí během půlroční lhůty, je z úložiště odstraněn. Pro specifikace očekávající oficiální zveřejnění se vžilo označení „work-in-progress“ (Bradner, 1996, s. 9). S výjimkou přímého adresování konceptu mohou být jako takové i odkazovány.

V případě dostatečného zájmu odborné veřejnosti je koncept specifikace nahrazen tzv. RFC dokumentem (Bradner, 1996, s. 9). Tento krok nicméně musí být doporučen skupinou IESG (Internet Engineering Steering Group). Ta má v rámci IETF na starost nejen standardizační proces, ale i technickou a administrativní správu celé organizace (Bradner, 1996, s. 12).

4.1 Requests for Comments

Požadavky ke komentáři jsou oficiálním kanálem pro publikaci platných i plánovaných internetových standardů. Je tomu tak již od roku 1969 a tedy od vzniku experimentální výzkumné sítě ARPANET (Bradner, 1996, s. 6). Odpovědnost za správu sérií RFC dokumentů nese projekt „RFC Editor“, který obstarává editaci, katalogizaci a publikování všech RFC dokumentů. Projekt odpovídá i za udržování závazných pravidel pro publikaci jednotlivých dokumentů, přičemž jedním z nich je i dostupnost přinejmenším v podobě ASCII textu (Bradner, 1996, s. 7).

RFC disponují jednoznačným identifikátorem a po své publikaci nejsou nikdy mazány. Jednotlivé dokumenty mohou být pouze doplněny jinými, případně jimi být označeny za zastaralé. Díky tomu jsou i překonané specifikace adresovatelné (Bradner, 1996, s. 8).

Samotné vydání RFC nicméně nevypovídá nic o standardizaci specifikace. Publikace v podobě RFC značí pouze dostatečný zájem odborné veřejnosti o danou specifikaci. Pokud specifikace naplní očekávání svých autorů a projde všemi stupni oficiálního procesu, pak má šanci stát se standardem. Standardy ve skutečnosti představují jednu z podmnožin RFC dokumentů a jejich souhrn je pravidelně zveřejňován ve speciálním RFC „Internet Official Protocol Standards” s označením STD 1 (Bradner, 1996, s. 7).

Proces standardizace kromě technických specifikací rozlišuje i tzv. prohlášení o použitelnosti (Applicability Statement). Tyto dokumenty popisují způsob použití jedné či kombinace více technických specifikací za účelem dosažení komplexní funkce. Mohou obsahovat nejen metody použití a kooperace, ale i provozní parametry specifikací (Bradner, 1996, s. 10).

Prohlášení o použitelnosti dělí jimi odkazované technické specifikace do tří skupin (Bradner, 1996, s. 10).

1. *Vyžadované* specifikace musejí být vždy implementované k dosažení alespoň minimálního souladu s prohlášením o použitelnosti.
2. *Doporučené* specifikace nemusejí být k dosažení souladu s prohlášením implementovány, nicméně jejich ignorování není dobrým zvykem. Jejich implementace je naopak často očekávána. Příkladem může být doporučení implementovat telnet všude tam, kde se očekává vzdálený přístup. Výrobci síťových prvků by doporučené specifikace měli ignorovat pouze za „speciálních podmínek”, které ale nejsou nijak definované.
3. *Volitelné* specifikace nejsou nijak vyžadovány ani očekávány. Jejich implementace tak závisí výhradně na dobré vůli daného výrobce.

Pro specifikace, které nejsou součástí tzv. „standards track” jsou zavedeny ještě dvě dodatečné úrovně (Bradner, 1996, s. 11).

1. *Limitované použití* značí nevhodnost jiného než experimentálního či jinak omezeného nasazení. Například specifikace experimentálního protokolu by měla být užívána výhradně k testování a dalšímu vývoji dané specifikace.
2. *Nedoporučené* jsou ty specifikace, které nejsou pro jejich stáří nebo vlastnosti vhodné pro široké nasazení.

4.2 Standards track

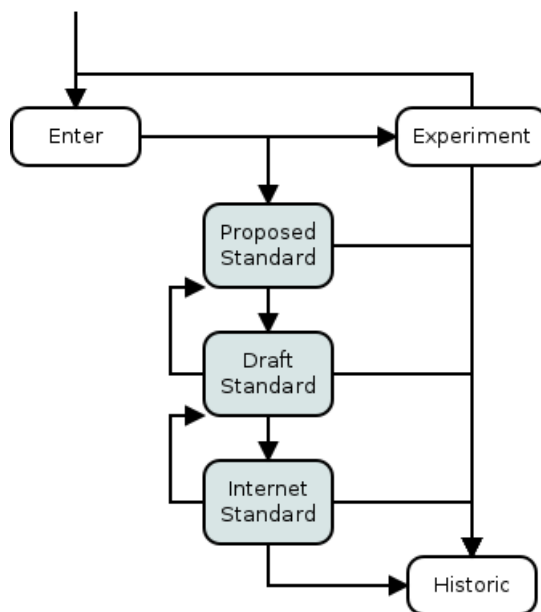
V rámci procesu standardizace prochází každá specifikace skupinou „úrovní vyspělosti“ (maturity levels). Každá z těchto úrovní klade na specifikaci různé požadavky, od pouhého zájmu příslušné komunity po technickou vyspělost a praktické nasazení. Přejedání mezi úrovněmi probíhá na základě „standardizační akce“ (standards action) vykonávané v reakci na rozhodnutí řídicí skupiny IESG. K takové akci a tedy k přechodu do další úrovně vyspělosti může u specifikace dojít i po její standardizaci (Bradner, 1996, s. 11). K tomu typicky dochází po objevu nových požadavků kladených na standardizovanou specifikaci v rámci zkušeností s jejím reálným použitím.

4.2.1 Standardizační akce

Přijetí specifikace do vstupní úrovně, její odebrání nebo povýšení do následující úrovně vyspělosti. To vše se souhlasem IESG a na doporučení příslušné pracovní skupiny v rámci organizace IETF. IESG provádí před každou standardizační akcí revizi požadavků kladených na specifikaci požadovanou úrovní vyspělosti. K tomuto kroku si může IESG vyžádat nezávislé technické posouzení internetovou komunitou (Bradner, 1996, s. 19). Pokud je povýšení specifikace v rámci „standards track“ schváleno, je o tomto rozhodnutí informován RFC Editor, který provede příslušné změny RFC a případné odebrání specifikace ze zvláštního úložiště konceptů (Bradner, 1996, s. 20).

4.2.2 Proposed Standard

Specifikace ve fázi „navrhovaného standardu“ by měla být především stabilní, alespoň co se jejích designových voleb týče. Odbornou veřejností by jí mělo být dobře rozuměno a měl by jí být přikládán dostatečný zájem a očekávání od jejího přínosu. Pokud se nejedná o specifikaci s přímým dopadem na funkci internetu, není v této úrovni vyžadována funkční implementace. Přesto je její existence žádoucí a může značně ovlivnit další rozhodování IESG při dalším přehodnocování specifikace. Navrhovaný standard by neměl trpět žádnými zřejmými nedostatky, přesto by měl být vývojáři považován za „nezralou“ specifikaci. Existence a praktické testovací nasazení může pomoci k odstranění chyb ve specifikaci a pomoci jí k povýšení do další úrovně (Bradner, 1996, s. 12).



Obrázek 17: Schéma úrovní vyspělosti

4.2.3 Draft Standard

Povýšení na „koncept standardu” je zásadním krokem v životním cyklu specifikace a musí mu předcházet existence alespoň dvou na sobě nezávislých implementací. Tyto implementace musí pocházet ze dvou nezávislých projektů, a pokud k jejich vývoji muselo být použito patentovaných technologií, jsou požadována dvě samostatná licenční řízení. Tento požadavek se vztahuje na všechny vlastnosti uvedené ve specifikaci.

Dojde-li tedy k situaci, kdy alespoň jedna ze dvou požadovaných implementací nedisponuje některou ze specifikovaných funkcí, pak tato funkce musí být doplněna do funkčního řešení, nebo odebrána ze specifikace. V opačném případě nemůže dojít k povýšení dané specifikace na koncept standardu. Koncept je považován za finální formu specifikace, v níž před standardizací dochází pouze k řešení drobných dílčích nedostatků (Bradner, 1996, s. 13). Samotné prohlášení za standard je tedy již spíše administrativním úkonem.

Proto se od konceptu očekává stabilní vývojová základna a podrobně zdokumentovaná historie testování všech specifikovaných funkcí. Od implementací vyžadovaných touto úrovní vyspělosti se očekává nejen maximální funkční spolehlivost, ale také vzájemná kompatibilita. Pokud spolu některé implementace nemohou spolupracovat, pak se vývojové týmy dopustili závažného odchýlení od specifikace. Případně může být tato specifikace nečitelná nebo chybná (Bradner, 1996, s. 13). K těmto odhalením ale typicky dochází ještě před povýšením na úroveň konceptu.

4.2.4 Internet Standard

Internetový standard nebo také pouze standard. Vyznačuje se vysokou úrovní technické vyspělosti, kterou prokázala svým rozšířením a praktickými zkušenostmi jejích uživatelů. Každému novému standardu je přiřazeno označení STD, zatímco si ponechává i předcházející RFC identifikátor (Bradner, 1996, s. 14).

4.2.5 Historic

Pokud byla specifikace na kterékoli úrovni vyspělosti překonána novější specifikací, respektive je-li z jakéhokoli důvodu považována za zastaralou, dostává se na „nestandardní“ historickou úroveň. Na historické specifikaci by neměl za normálních okolností záviset žádný z platných standardů. Obdobně by žádná specifikace neměla záviset na specifikaci nacházející se na nižší úrovni vyspělosti (Bradner, 1996, s. 16).

4.2.6 Experimental

Experimentální specifikace slouží jako dokumentace aktivního výzkumu a účelem jejího publikování je informovat odbornou veřejnost o aktuálním stavu a dalším vývoji, případně o jeho koordinaci. Autorem těchto specifikací mohou být nejen vývojové a pracovní skupiny IETF, ale i týmy či jedinci z vnějšku této organizace (Bradner, 1996, s. 14).

4.2.7 Nestandardní úrovně vyspělosti

Je očividné, že ne všechny úrovně musí nutně vést ke standardizaci. Specifikace na těchto úrovních nejsou dostatečně rozvinuté, nebo jsou již naopak překonány jinými (Bradner, 1996, s. 14). Do této skupiny spadají úrovně historická, experimentální, ale také tzv. *informativní* úroveň vyspělosti.

Informational

Informativní specifikace slouží k publikování všeobecných dokumentů nesoucích široký obsah informací z mnoha zdrojů stojících například i zcela mimo standardizační proces. Jedním z účelů těchto dokumentů je i koordinační činnost napříč nejenom vývojovými týmy v IETF i mimo ni (Bradner, 1996, s. 15).

4.3 Standardizační proces

Proces standardizace se skládá z řady rozhodnutí řídicí skupiny IESG ovlivňující přijetí specifikace do vstupní úrovně „standards track” a její povýšení do úrovně následující. Případně zrušení standardizace dané specifikace a tedy její odebrání z libovolné úrovně.

Ačkoli je mechanismus průchodu těmito úrovněmi zřejmý, pro vývojáře a autory specifikací neexistují žádné záruky přijetí, povýšení a konečné standardizace jejich specifikace (Bradner, 1996, s. 18). Prvním krokem procesu je samotné přijetí specifikace to „standards track”, a tedy vykonání příslušné „standardizační akce” řídicí skupinou IESG.

4.3.1 Vyvolání standardizační akce

Ačkoli RFC tento krok explicitně zmiňuje, zahrnuje pouze podání doporučení pracovní skupinou IETF nebo jedincem. Pracovní skupina podává doporučení svému řediteli, jedinec stojící za specifikací nepřirazené žádné skupině podává doporučení přímo IESG (Bradner, 1996, s. 18).

4.3.2 Revize a schválení akce

V reakci na doporučení pracovní skupiny či jedince se řídicí skupina IESG začne zabývat tím, zda příslušná specifikace splňuje požadovaná kritéria a formální požadavky pro povýšení do požadované úrovně vyspělosti. V závislosti na důležitosti specifikace si může IESG vyžádat nezávislou revizi specifikace. K tomu dochází především u specifikací s přímým dopadem na funkci internetu.

Schválení akce předchází zaslání upozornění elektronickou poštou. Toto upozornění je zasláno do veřejně dostupné elektronické diskuse, kde se musí nacházet po určitou dobu, během níž má odborná veřejnost možnost revidovat danou specifikaci. U specifikací, za něž odpovídá pracovní skupina IETF, odpovídá tato doba minimálně dvěma týdnům. U externích specifikací spravovaných jedincem je prodloužena na minimálně čtyři týdny.

Klíčem pro schválení standardizační akce je nezávislost řídicí skupiny IESG. Ta není doporučením akce nijak vázána a jak s ním naloží, závisí čistě na členech IESG a jejich konsenzu.

IESG má dále možnost zveřejnit specifikaci v jiné než požadované kategorii, popřípadě ustanovit jinou pracovní skupinu zabývající se danou specifikací. K tomu typicky dochází v reakci na kontroverzi, kterou může posuzování specifikace odbornou veřejností vyvolat (Bradner, 1996, s. 18-19).

4.3.3 Zveřejnění

Po schválení standardizační akce přichází na řadu její zveřejnění RFC Editorem. Pokud je zveřejňována specifikace, jejíž koncept se nachází ve zvláštním úložišti konceptů, pak je z toho úložiště odstraněn současně s uveřejněním příslušného RFC dokumentu. Pokud to standardizační akce vyžaduje, je RFC Editor povinen zveřejnit aktualizovaný dokument STD 1, tedy seznam mapující platné internetové standardy (Bradner, 1996, s. 19-20).

4.3.4 Selhání procesu

Selhání standardizačního procesu je definováno jako individuální nesouhlas s rozhodnutím řídicí skupiny IESG. Osoba vyjadřující tento nesouhlas se nejdříve musí obrátit na předsedajícího člena skupiny IESG. Pokud ten nedokáže problém uspokojivě vyřešit, musí se jím zabývat celá skupina. IESG se poté zabývá nejen problémovým rozporem, ale také akcí, která jeho vzniku předcházela. Výsledkem je rozhodnutí o případném vykonání další akce řešící tento rozpor, nebo pouhá zpráva o urovnání sporu.

Pokud IESG nedokáže spor urovnat, vkládá se do řešení architektonická rada IAB (Internet Architecture Board). V moci rady je vydat příslušné doporučení IESG k řešení sporu, případně anulovat její předešlou akci. Rozhodnutí rady je finální a nelze ho zvrátit. IESG je poté povinna přehodnotit své předcházející „problémové” rozhodnutí tak, jako by k němu nedošlo. Cílem je tedy přijmout takové opatření, které nevyvolá další spor (Bradner, 1996, s. 23).

4.4 Revidování standardu

Každá zásadní revize zavedeného standardu, která je postupem času transformována do nové verze protokolu, musí projít kompletním standardizačním cyklem tak, jako by se jednalo o zcela novou a nezávislou specifikaci.

Prohlášení nové verze protokolu za standard typicky znamená nahrazení původního standardu novým. Pro starý standard to tedy znamená ukončení jeho platnosti neboli tzv. „odchod do důchodu“. Má-li předešlá verze standardizovaného protokolu dostatečně širokou uživatelskou základnu, poté mohou zůstat v platnosti obě verze protokolu.

Vztah obou standardů nicméně musí být jasně uveden v RFC dokumentu nového standardu. Dalším důvodem pro zpoplatnění obou verzí téhož protokolu může být příliš náročný proces přechodu na nový standard, a to především z pohledu národních síťových infrastruktur a velkých hráčů z oboru síťového průmyslu (Bradner, 1996, s. 21).

Pokud to ale situace dovolí, nebo je nový standard technologicky natolik nadřazen svému předchůdci, pak je původní standard prohlášen za zastaralý a získává status historické specifikace. Přiřazení tohoto označení zastaralé specifikaci prochází stejným schvalovacím procesem jako kterákoli jiná standardizační akce. Požadavek na zastarání protokolu může pocházet od pracovní skupiny, jejího ředitele nebo zainteresované třetí strany (Bradner, 1996, s. 21).

4.5 Standardizační authority

Procesem standardizace síťových protokolů se nezabývá pouze IETF, po celém světě existuje řada národních i nadnárodních standardizačních agentur, které se zabývají specifikací mnoha standardů včetně těch popisujících síťovou komunikaci. Vlastní tzv. proprietární standardy si definuje i řada soukromých společností z oboru informačních technologií.

IETF z pozice organizace zodpovídající za funkci a rozvoj internetu dělí externí protokoly do dvou skupin (Bradner, 1996, s. 24).

4.5.1 Proprietární specifikace

Tyto protokoly jsou pevně pod kontrolou svých tvůrců, kterými jsou soukromé společnosti. Jejich nasazení je typicky omezeno na modelové řady konkrétního výrobce, nicméně mohou být i licencovány třetím stranám. IETF k těmto specifikacím může přistupovat jako ke standardům, pokud jsou v průmyslu dostatečně rozšířeny. Řada výrobců se účelově snaží své protokoly s různou mírou úspěchu a otevřenosti standardizovat (Bradner, 1996, s. 24).

4.5.2 Otevřené standardy

Druhou skupinou jsou tzv. otevřené standardy z dílen rozličných standardizačních organizací. Tyto organizace zveřejňují své specifikace v obdobné formě jako IETF, ta je proto považuje za „otevřené externí standardy” (Bradner, 1996, s. 25).

ANSI

Americký národní standardizační institut (American National Standards Institute) byl založen již v roce 1918 skupinou soukromých i vládních subjektů. Jedná se o soukromou neziskovou organizaci sdružující privátní a vládní objekty ve snaze standardizovat široce užívané technologie. Členství v této organizaci i dodržování jejích standardů je zcela dobrovolné. Samotný ANSI institut je členem mnoha jiných standardizačních agentur, díky čemuž může propagovat americké standardy na mezinárodní scéně a tak podporovat jejich rozšíření (ANSI, 2017).

ISO

ISO je zkratkou pro „International Organization for Standardization”, nevládní mezinárodní agenturu se sídlem ve Švýcarsku. ISO byla založena počátkem roku 1947 v reakci na snahu 25 zemí vytvořit organizaci koordinující průmyslovou standardizaci. Členy této organizace jsou dnes subjekty z více než 160 zemí a počet vydaných standardů se vyšplhal na 21686 (ISO, 2017).

IEEE

Institut IEEE (Institute of Electrical and Electronics Engineers) vznikl v roce 1963 sloučením dvou jiných amerických agentur a v současnosti má přes 420 000 členů z více než 160 zemí světa. Kromě standardizace se IEEE zabývá také vzdělávací činností a pořádáním technologických konferencí (IEEE, 2017).

ITU-T

Pod záštitou organizace Spojených Národů (United Nations) funguje standardizační agentura ITU-T, která tvoří jeden ze tří pilířů mezinárodní telegrafní unie (ITU). Vznik unie se datuje do roku 1865, nicméně agentura ITU-T vznikla až v roce 1993 spojením a přejmenováním dvou komisí fungujících v rámci ITU. Agentura se soustředí na standardizaci technologií v oboru telekomunikačních sítí a své standardy vydává ve formě tzv. doporučení (ITU-T Recommendations) (ITU, 2017).

5 Demonstrační implementace

Za účelem kontroly funkce byla během vývoje protokolu ESP vytvořena jeho testovací implementace. Vytvořenou aplikaci je možné použít pro detailní monitorování všech akcí definovaných v protokolu, jakož i modulů a funkcí tyto akce implementujících. Ukázková aplikace disponuje přehledným grafickým prostředím, díky kterému ji lze velice snadno ovládat. Na svou obsluhu okno aplikace klade pouze minimální uživatelské nároky.

5.1 Vývojové knihovny

Při vývoji implementace ESP protokolu byl kladen důraz především na nízké systémové požadavky a kompatibilitu napříč nejznámějšími operačními systémy. Cílovou platformou protokolu byly na prvním místě různorodé linuxové distribuce následované operačními systémy z rodiny MS Windows. Proto byl pro vývoj aplikace zvolen dobře známý aplikační rámec (framework) pro vývoj multiplatformních aplikací Qt páté generace zahrnující podporu standardu C++11.



Obrázek 18: Okno aplikace

Aplikaci lze sestavit pomocí Qt 5.6 s dlouhodobou podporou LTS (Long Term Support), nicméně pro její vývoj byly použity zejména knihovny

- Qt 5.3.2 se stále trvající prodlouženou podporou
- a Qt 5.7.1, jejichž prodloužená podpora započala 16.6.2017.

5.1.1 Vývoj a kompatibilita

Díky multiplatformnosti a značnému rozšíření knihoven Qt nevznikla během vývoje implementace potřeba pro dualitu zdrojového kódu v závislosti na cílovém systému. Přesto bylo nutné věnovat čas studiu dokumentace funkcí knihoven Qt, a to především z důvodu včasného nalezení známých problémů nebo nestandardního chování vybraných funkcí. Jako příklad lze uvést metodu „QAbstractSocket::waitForBytesWritten()”, která dle dokumentace náhodně selhává při použití na operačních systémech Windows.

```
bool QAbstractSocket::  
waitForBytesWritten(int msecs = 30000)
```

This function blocks until at least one byte has been written on the socket and the `bytesWritten()` signal has been emitted. The function will timeout after *msecs* milliseconds; the default timeout is 30000 milliseconds.

The function returns `true` if the `bytesWritten()` signal is emitted; otherwise it returns `false` (if an error occurred or the operation timed out).

Note: This function may fail randomly on Windows. Consider using the event loop and the `bytesWritten()` signal if your software will run on Windows.

Obrázek 19: Dokumentace funkce (Qt, 2017)

Podrobný popis celého zdrojového kódu je nad rámec této práce. Všechny zdrojové soubory doplněné o dokumentační komentáře jsou k této práci přiloženy na datovém nosiči.

5.1.2 Struktura aplikace

Aplikace je funkcionálně rozdělena do tří téměř samostatných modulů. Každý z nich je deklarován v samostatném hlavičkovém souboru a za účelem odpovědného plnění svých povinností spolu úzce spolupracují.

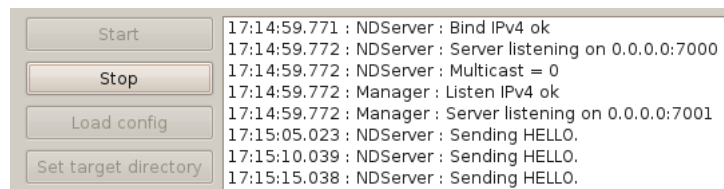
1. Prvním z těchto modulů je modul objevování sousedů popsáný v hlavičkovém souboru NDserver.h. Tento modul je zodpovědný výhradně za zpracování HELLO zpráv, které v závislosti na jejich časovači odesílá na příslušnou skupinovou adresu. Dále také přijímá HELLO zprávy zasílané ostatními uzly na k tomu určený UDP port, přičemž tyto zprávy předává ke zpracování následujícímu modulu.
2. Druhý modul je deklarovaný v souboru Manager.h a má na starosti udržování a správu kolekce uzlů. Modul obstarává předávání povelů přicházejících z grafické nadstavby a tedy potažmo od samotného uživatele. Přes tohoto „správce“ plynou také informace o přijatých HELLO zprávách, v důsledku čehož dochází k resetování příslušných časovačů odpovídajících uzlů.
3. Posledním modulem jsou uzly respektive jejich kolekce. Ty jsou deklarovány v souboru Node.h a představují největší strukturu v celém programu. Každý uzel zastupuje instanci tohoto programu běžící na vzdáleném počítači a obsahuje zejména funkcionalitu obstarávající přenos a správu sdílených souborů. Zatímco většina programu pracuje ve výchozím vlákně aplikace, uzly delegují náročnější operace do k tomu určených vláken.

Oproti zavedeným protokolům a jejich nejznámějším implementacím se jedná o poměrně jednoduchý návrh. I přes inspiraci jinými známými technologiemi si návrh zachovává odstup od protokolů popsáných v předešlých kapitolách a snaží se najít nový střízlivý pohled na definici uživatelské přívětivosti, a to především absencí zdlouhavé konfigurace před prvním spuštěním aplikace.

5.2 Ovládání aplikace

5.2.1 Spuštění aplikace

Aplikace zahajuje svou činnost po stisknutí tlačítka start. V reakci na to začíná program naslouchat na portech 7000, na něž jsou zasílány HELLO zprávy, a 7001, na který jsou zasílány požadavky o navázání spolehlivého spojení. O průběhu svázání aplikace s danými porty je uživatel informován výpisem logu. Pokud bylo svázání s portem 7000 úspěšné, začíná program s periodickým zasíláním HELLO zpráv na příslušnou skupinovou adresu.

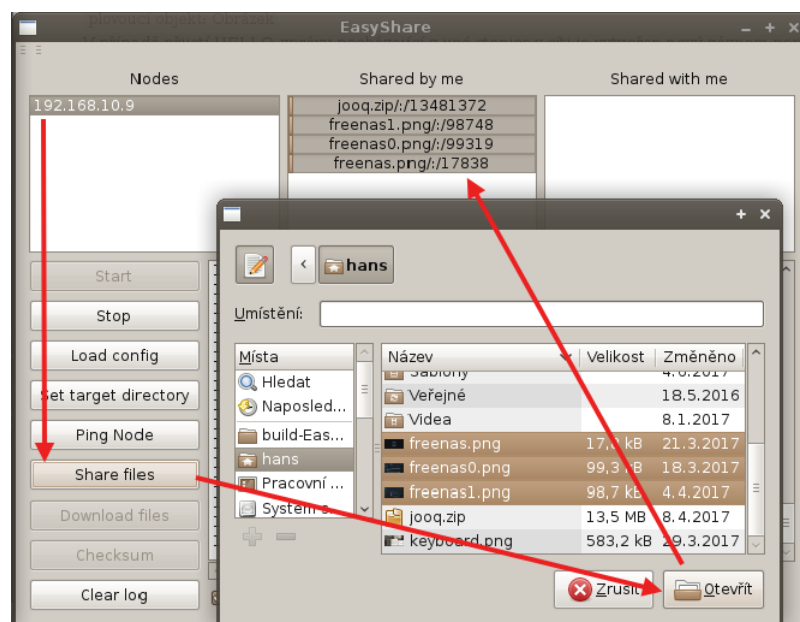


Obrázek 20: Start aplikace

V případě přijetí HELLO zprávy pocházející z jiné stanice v síti je vytvořen nový záznam popisující vzdálený uzel a tato akce je indikována přidáním jeho síťové adresy do seznamu uzlů. Po jeho označení levým tlačítkem myši jsou uživateli zpřístupněny akce uzlu. Výpis logu může také zobrazovat informaci o existenci konfigurace pro příslušný uzel. Ta se nachází v XML souboru v podadresáři „EasyShare” domovské složky uživatele. Konfigurace obsahuje seznam souborů, které byly v minulosti sdíleny s daným vzdáleným uzlem. Soubor lze načíst tlačítkem „Load Config”.

5.2.2 Sdílení souborů

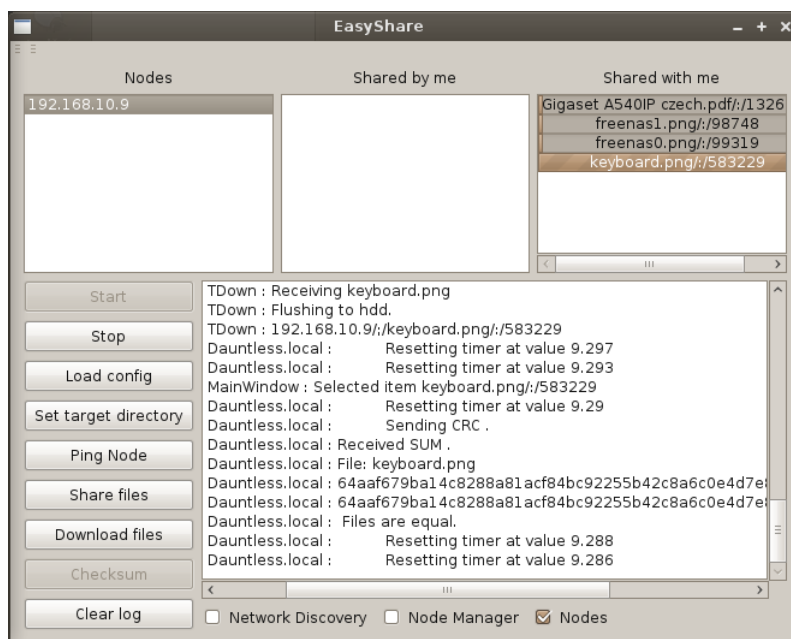
Jednou z akcí uzlu je sdílení souborů. Po stisknutí tlačítka „Share” je zobrazeno okno pro výběr jednoho nebo více souborů. Po úspěšném výběru jsou jejich názvy a velikost zobrazeny v seznamu uzlem sdílených souborů. Seznam je v korektním formátu okamžitě odeslán příslušnému uzlu v síti. Na protější straně spojení jsou odeslané informace zobrazeny v seznamu „s uzlem sdílených souborů”.



Obrázek 21: Sdílení souborů

5.2.3 Stažení souboru

Stažení protistranou sdílených souborů probíhá obdobným způsobem. Uživatel nejdříve musí zvolit jeden ze souborů zobrazených v seznamu s ním sdílených souborů, v reakci na což je mu umožněno soubor stáhnout. Cílové umístění, kam má být soubor uložen, lze předem upravit pomocí tlačítka „Set target directory”. Po stažení souboru lze ověřit jeho pravost výpočtem kontrolních součtů. K tomu slouží tlačítko „Checksum” a o výsledku těchto výpočtů je uživatel informován ve výpisu logu.



Obrázek 22: Kontrolní součty souboru

6 Závěr

V této práci jsem se zabýval procesem návrhu a následné implementace síťového protokolu pro sdílení souborů v lokální síti. Obor projektování síťových technologií si jistým způsobem zachovává stigma vysoké náročnosti. Samotný termín „síťový protokol“ často navozuje dojem jakési černé skříňky, o jejíž funkci nemá běžný uživatel nejmenší tušení. Pokročilý uživatel zase musí věnovat dlouhé hodiny nebo i celé dny studiu vlastností, pravidel, drobností a všemožných zákonitostí daného protokolu, aby se z něj stal odborník na slovo vzatý. Se znalostí protokolu jde ale ruku v ruce znalost alespoň těch nejznámějších z jeho implementací, protože znalost jednoho je bez znalosti druhého často téměř bezcenná.

Představa o nezdolné náročnosti návrhu síťového protokolu se ale může začít hroutit, zaměříme-li se na samotnou definici protokolu. Tu lze snadno zjednodušit na prostou množinu příkazů a jim odpovídající množinu následných akcí. V tomto podání lze pojem protokol uplatnit v podstatě na jakoukoli činnost vykonávanou téměř každý den v měsíci. Pokud se určitým protokolem může řídit kdokoli, nemělo by být obtížné definovat svůj protokol pro kohokoli. Toto tvrzení staví proces návrhu protokolu do zcela jiného světla, nicméně od obecného protokolu je to stále kus cesty k protokolu síťovému. Ten i přes všechna zjednodušení vyžaduje určitou míru znalostí nejen z oboru síťových technologií.

Jak naznačuje Gerard J. Holzmann ve své knize, návrh protokolu prochází během svého vývoje mnoha změnami, které ne vždy vedou k úspěchu. První pokusy o implementaci nového protokolu často vedou k objevu dosud neznámých chyb a zjevných nedostatků. Tento prvotní neúspěch často vede k přehodnocení představ o principech a funkci nového protokolu a může vést i k jeho kompletnímu přepracování. Výjimkou nebyl ani návrh protokolu popsaného v této práci. Praktické nasazení nicméně dokázalo, že protokol dostal přinejmenším základním požadavkům, které na něj byly od počátku kladeny.

Protokol ESP je založen na poměrně jednoduchém způsobu zasílání příkazů, který zcela jistě není ideální. Při množství existujících technologických postupů a pohledů na tuto problematiku je ale v podstatě nemožné označit jeden z nich za ideální. Nejednou bylo dokázáno, že méně může být více. I s potenciálně minimálním rozšířením výsledku takovéto práce je proces návrhu a implementace vlastního protokolu hodnotnou zkušeností.

Dokončení této práce nicméně nemusí nutně znamenat konec práce na zde popsaném protokolu. Inspirací může být protokol NFS, jehož první verzi vývojáři prakticky nepředstavili světu. Namísto toho po jeho „kompletaci“ pokračovali v práci, na jejímž konci bylo veřejné vydání hned druhé verze jejich protokolu. Takových vydání může být představeno takové množství, že určitou inspiraci lze najít i v práci vývojářů společnosti Microsoft. Tedy vydat tolik nových verzí, že si již málokdo vzpomene na to, kde a jak vznikl originální protokol.

1. TRIDGELL, A. Myths About Samba [online]. 2005 [cit. 2017-01-28].
Dostupné z: <https://www.samba.org/samba/docs/myths_about_samba.html>.
2. SHARPE, R. Just what is SMB? [online]. c1996, poslední revize Oct 2002 [cit. 2017-02-01]. Dostupné z: <<https://www.samba.org/cifs/docs/what-is-smb.html>>.
3. TRIDGELL, A. et al. A bit of history and a bit of fun [online]. c1997, Oct 1998 [cit. 2017-01-30]. <<http://www.rxn.com/services/faq/smb/samba.history.txt>>.
4. TRIDGELL, A. Efficient Algorithms for Sorting and Synchronization. Canberra, Feb 1999. A thesis submitted for the degree of Doctor of Philosophy at The Australian National University.
5. BARRETO, J. SMB2, a complete redesign of the main remote file protocol for Windows. *TechNet* [online]. Dec 2008 [cit. 2017-02-02]. Dostupné z <<https://blogs.technet.microsoft.com/josebda/2008/12/09/smb2-a-complete-redesign-of-the-main-remote-file-protocol-for-windows/>>.
6. EVANS D., T. NetBIOS, NetBEUI, NBF, NBT, NBIPX, SMB, CIFS Networking [online]. Mar 1987, poslední revize 2003 [cit. 2017-02-06]. Dostupné z <<http://timothydevans.me.uk/nbf2cifs/nbf2cifs.pdf>>. ISSN: 2070-1721.
7. BECK, M. Virtualization Acceleration. In: INTERNET SOCIETY. [online]. Sep 2014 [cit. 2017-02-01]. Dostupné z <<http://www.slideshare.net/mellanox/motti-virtualization-accelration-over-ro-ce>>.
8. MICROSOFT Corp. Server Message Block (SMB) Protocol Versions 2 and 3. *Microsoft API and reference catalog* [online]. Jun 2017, [cit. 2017-02-08]. Dostupné z <<https://msdn.microsoft.com/en-us/library/cc246482.aspx>>.
9. SANDBERG, R. et al. 1985 The Sun Network Filesystem: Design, Implementation and Experience. *Technical report*. Jul 1985, [cit. 2017-03-10]. Dostupné z: <http://web.mit.edu/6.033/2002/wwwdocs/papers/nfs.pdf>
10. Sun Microsystems, Inc. RPC: Remote Procedure Call Protocol Specification Version 2. *Request for Comments: 1057* [online]. Jun 1988, [cit. 2017-03-03]. Dostupné z <<https://tools.ietf.org/html/rfc1057>>. ISSN: 2070-1721.

11. SRINIVASAN, R. XDR: External Data Representation Standard. *Request for Comments: 1832* [online]. Aug 1995, [cit. 2017-03-11]. Dostupné z <<https://tools.ietf.org/html/rfc1832>>. ISSN: 2070-1721.
12. REYNOLDS, J. et al. ASSIGNED NUMBERS. *Request for Comments: 1340* [online]. Jul 1992, [cit. 2017-03-12]. Dostupné z: <<https://tools.ietf.org/html/rfc1340>>. ISSN: 2070-1721.
13. TOUCH, J. et al. Service Name and Transport Protocol Port Number Registry [online]. c2017, poslední revize 26.7.2017 [cit. 2017-03-28]. Dostupné z: <<https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>>.
14. SHEPLER, S. Remote Procedure Call (RPC) Program Numbers [online]. c2009, poslední revize 30.3.2015 [cit. 2017-03-20]. Dostupné z: <<https://www.iana.org/assignments/rpc-program-numbers/rpc-program-numbers.xhtml>>.
15. MAITI, K. How RPC works? [online]. c2011 [cit. 2017-03-02]. Dostupné z <<http://kmaiti.blogspot.cz/2011/03/how-rpc-works.html>>.
16. PETRON, E. Remote Procedure Calls. Linux Journal [online]. Oct 1997 [cit. 2017-03-18]. Dostupné z <<http://www.linuxjournal.com/article/2204>>.
17. Sun Microsystems, Inc. NFS: Network File System Protocol Specification. *Request for Comments: 1094* [online]. Mar 1989, [cit. 2017-03-10]. Dostupné z <<https://tools.ietf.org/html/rfc1094>>. ISSN: 2070-1721.
18. CALLAGHAN, B. et al. NFS Version 3 Protocol Specification. *Request for Comments: 1813* [online]. Jun 1995, [cit. 2017-03-18]. Dostupné z <<https://tools.ietf.org/html/rfc1813>>. ISSN: 2070-1721.
19. RAADT, T. Re: nfsv4? In: mailing list misc@openbsd.org. Oct 27. 2010. [cit. 2017-03-12]. <<http://openbsd-archive.7691.n7.nabble.com/nfsv4-td18690.html>>.
20. SHEPLER, S. et al. Network File System (NFS) Version 4 Protocol. *Request for Comments: 3530* [online]. Apr 2003, [cit. 2017-03-19]. Dostupné z: <<https://tools.ietf.org/html/rfc3530>>. ISSN: 2070-1721.

21. HAYNES, Ed. T. et al. Network File System (NFS) Version 4 Protocol. *Request for Comments: 7530* [online]. Mar 2015, [cit. 2017-03-25]. Dostupné z: <<https://tools.ietf.org/html/rfc7530>>. ISSN: 2070-1721.
22. SHEPLER, S. et al. Network File System (NFS) Version 4 Minor Version 1 Protocol. *Request for Comments: 5661* [online]. Jan 2010, [cit. 2017-03-20]. Dostupné z: <<https://tools.ietf.org/html/rfc5661>>. ISSN: 2070-1721.
23. ZHANG, S. pNFS Introduction. In: SlideShare [online]. Nov 2009 [cit. 2017-03-10]. Dostupné z: <<https://www.slideshare.net/schubertzhang/pnfs-introduction>>.
24. KERRISK, M. nfs - fstab format and options for the nfs file systems. In: Linux man-pages [online]. Oct 2012 [cit. 2017-03-09]. Dostupné z <<http://man7.org/linux/man-pages/man5/nfs.5.html>>.
25. BRADNER, S. The Internet Standards Process – Revision 3. *Request for Comments: 2026* [online]. Oct 1996, [cit. 2017-04-01]. Dostupné z <<https://tools.ietf.org/html/rfc2026>>. ISSN: 2070-1721.
26. American National Standards Institute (ANSI). About ANSI [online]. c2017 [cit. 2017-07-14]. <https://www.ansi.org/about_ansi/overview/overview>.
27. International Organization for Standardization (ISO). About ISO [online]. c2017 [cit. 2017-07-15]. <https://www.ansi.org/about_ansi/overview/overview>.
28. Institute of Electrical and Electronics Engineers (IEEE). IEEE at a Glance [online]. c2017 [cit. 2017-07-15]. <https://www.ieee.org/about/today/at_a_glance.html>.
29. International Telecommunication Union (ITU). ITU-T in brief [online]. c2017 [cit. 2017-07-16]. <<http://www.itu.int/en/ITU-T/about/Pages/default.aspx>>.
30. MOY, J. OSPF Version 2. *Request for Comments: 1583* [online]. Mar 1994, [cit. 2017-06-12]. Dostupné z <<https://tools.ietf.org/html/rfc1583>>. ISSN: 2070-1721.
31. Qt Documentation [online]. c2017, poslední revize 2017 [cit.2017-06-15]. Dostupné z: <<http://doc.qt.io/qt-5/qabstractsocket.html#waitForBytesWritten>>.