

Zhodnocení výkonnosti různých typů komunikace u distribuovaných aplikací

Performance Considerations for Different Communications Used
in Distributed Applications

Bc. Tomáš Kusák

Diplomová práce
2013



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Tomáš Kusák**
Osobní číslo: **A11405**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **prezenční**

Téma práce: **Zhodnocení výkonnosti různých typů komunikace u distribuovaných aplikací**

Zásady pro vypracování:

1. Vypracujte literární rešerši na téma možnosti komunikace u distribuovaných aplikací v prostředí .NET (WCF, Web API a dalších).
2. Navrhněte testovací prostředí a vytvořte software v prostředí .NET, který umožní měřit rychlost přenosu dat a další charakteristiky.
3. Provedte praktické testy přenášení různých vzorků dat přes různé technologie s různým nastavením.
4. Naměřené hodnoty přehledně zpracujte, srovnajte technologie z hlediska režie při přenosu.
5. Z naměřených dat vyvodte obecná doporučení o vhodnosti použití jednotlivých technologií v závislosti na požadovaném způsobu nasazení.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. SHARP, John. Windows Communication Foundation 4 step by step. Sebastopol, California.: Microsoft/O'Reilly, 2010, 700 s. ISBN 07-356-4556-6.
2. GALLOWAY, Jon Alan. Professional asp.net mvc 4. 1st ed. Indianapolis, Indiana: Wiley Publishing Inc., 2012, 432 s. ISBN 11-183-4846-X.
3. KOZIEROK, Charles M. The TCP/IP guide: a comprehensive, illustrated Internet protocols reference. San Francisco: No Starch Press, 2005, 1616 s. ISBN 15-932-7047-X.
4. Pro C-Sharp and the .NET 4.5 framework. 6. edition. Berkeley, California: APress., 2012, 1560 s. ISBN 978-143-0242-338.
5. LAMPING, Ulf, Richard SHARPE a Ed WARNICKE. Wireshark User's Guide: for Wireshark 1.9 [online]. 2012 [cit. 2013-01-27]. Dostupné z: <http://www.wireshark.org/download/docs/user-guide-a4.pdf>.
6. MSDN magazine [online]. 2012, vol. 27, no. 12 [cit. 2013-01-27]. ISSN 1528-4859. Dostupné z: http://download.microsoft.com/download/6/3/E/63EEE0EA-D7A8-4BE6-B71C-5E3DFDCE18F9/MDN_1212DG.pdf.

Vedoucí diplomové práce:

Ing. Jiří Pálka, Ph.D.

Ústav elektroniky a měření

Datum zadání diplomové práce:

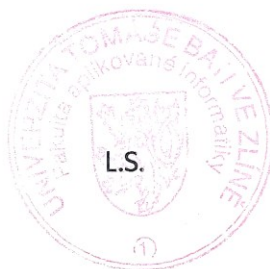
22. února 2013

Termín odevzdání diplomové práce:

22. května 2013

Ve Zlíně dne 22. února 2013

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Tato diplomová práce se zabývá praktickým testováním výkonnosti různých způsobů síťové komunikace a přenosu dat na platformě .NET.

V teoretické části je rozebráno, co to je distribuovaná aplikace, jakými způsoby může komunikace probíhat, jaké datové formáty se pro komunikaci používají a jaké technologie lze k tomuto účelu použít na platformě .NET.

Praktická část se zabývá testováním vybraných způsobů komunikace technologie WebAPI a WCF z hlediska trvání volání funkce při přenosu různého typu a množství dat a velikosti režie při přenosu. Testování probíhalo v laboratorních i veřejných podmínkách. Naměřená data jsou statisticky zpracována a na základě výsledků jsou dle klíčových požadavků doporučeny jednotlivé technologie, protokoly a jejich nastavení.

Klíčová slova: Microsoft, .NET, WebAPI, WCF, REST, SOAP, síť, komunikace, měření

ABSTRACT

This thesis is concerned about practical performance tests of different ways of network communication and data transfer on .NET platform.

In theoretical part, there is examined what is distributed application, in which ways could be communication realized, which data formats are used for communication and which technologies are available for this purpose on .NET platform.

Practical part is about testing selected ways of communication of WebAPI and WCF technologies from the point of time which is needed to call function which is transferring data of different types and sizes by given way and point of overhead of each way. Tests have been done in laboratory and public conditions. Recorded data have been statistically processed and based on results and key requirements, there are recommendations for particular technologies, protocols and settings.

Keywords: Microsoft, .NET, WebAPI, WCF, REST, SOAP, network, communication, measurement

Děkuji svému vedoucímu diplomové práce Ing. Jiřímu Pálkovi, Ph.D. za to, že mi ochotně věnoval spoustu svého času a umožnil mi získat praktické zkušenosti v oblasti programování a síťové komunikace.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně dne 29. dubna 2013

.....
podpis diplomanta

OBSAH

ÚVOD	11
I TEORETICKÁ ČÁST	12
1 ÚVOD DO DISTRIBUOVANÝCH SYSTÉMŮ	13
1.1 CO JE TO DISTRIBUOVANÝ SYSTÉM	13
1.2 K ČEMU JE DISTRIBUOVANÝ SYSTÉM DOBRÝ.....	13
1.2.1 Sdílení výkonu.....	13
1.2.2 Distribuce dat	14
1.2.3 Rozdělení funkcionality na jednotlivé uzly.....	15
1.3 DISTRIBUOVANÉ APLIKACE.....	16
1.3.1 Komunikace v distribuovaných aplikacích	16
2 BĚHOVÉ PROSTŘEDÍ .NET	18
2.1 SOUČÁSTI .NET	18
2.2 KOMPILACE	19
2.3 TECHNOLOGIE POSTAVENÉ NA PROSTŘEDÍ .NET	20
2.3.1 Windows Forms	20
2.3.2 WPF.....	20
2.3.3 ASP.NET.....	20
2.3.4 WCF	20
2.3.5 LINQ	20
2.3.6 WF.....	20
3 DATOVÉ FORMÁTY POUŽÍVANÉ PŘI PŘENOSU	22
3.1 XML.....	22
3.1.1 Základní formát jazyka XML.....	23
3.1.2 Definice typu dokumentu	24
3.1.3 Jmenné prostory	25
3.1.4 XML schéma	26
3.2 JSON	27
3.2.1 Formát jazyka JSON	27
3.3 POROVNÁNÍ XML A JSON.....	28

4	SÍŤOVÁ KOMUNIKACE NA NIŽŠÍCH VRSTVÁCH.....	29
4.1	TCP (TRANSMISSION CONTROL PROTOCOL)	29
4.1.1	Funkce protokolu TCP	30
4.1.2	Režie protokolu TCP	31
4.2	IP (INTERNET PROTOCOL).....	32
4.2.1	Režie protokolu IP	33
4.3	ETHERNET (IEEE 802.3).....	34
4.3.1	Režie protokolu Ethernet 802.3ab	35
5	ZPŮSOBY KOMUNIKACE	36
5.1	REST	36
5.2	PROTOKOL HTTP – NEJČASTĚJŠÍ IMPLEMENTACE PRINCIPU REST.....	37
5.2.1	Dotaz na server	37
5.2.2	Odpověď serveru	39
5.2.3	HTTPS.....	39
5.3	SOAP	40
5.3.1	Přenos binárních dat	41
5.3.2	Vzdálené volání procedur	42
5.3.3	Rozšíření WS-*	45
5.3.4	WSDL.....	46
6	DISTRIBUOVANÉ APLIKACE V .NET	47
6.1	WEBAPI	47
6.1.1	Mapování URL adresy na metodu WebAPI.....	47
6.1.2	Rozšiřitelnost.....	49
6.2	WCF.....	50
6.2.1	Interface.....	50
6.2.2	Implementace služby	51
6.2.3	Komunikační bod	52
6.2.4	Binding	53
6.3	WEBSOCKET	54
7	PROSTŘEDKY PRO ANALÝZU A ÚPRAVU SÍŤOVÉ KOMUNIKACE.....	57
7.1	WIRESHARK.....	57
7.2	CASCADE PILOT	58
7.3	NETLIMITER	59

II	PRAKTICKÁ ČÁST	61
8	ZÁMĚRY TESTU	62
8.1	ZÁKLADNÍ ROZDĚLENÍ TESTŮ	63
8.1.1	Test vlivu serializace objektu do formátů JSON, XML, SOAP	63
8.1.2	Test vlivu použité technologie a formátu na dobu stahování binárních dat.....	63
8.1.3	Test vlivu různých možností při uploadu souboru na server	63
9	ZPŮSOB PROVEDENÍ TESTŮ	64
9.1	TESTY STAHOVÁNÍ DAT	64
9.1.1	Přehled přenášených jednotek dat	65
9.1.2	Přehled způsobů přenosu dat.....	66
9.2	TESTY ODESÍLÁNÍ DAT	66
9.2.1	Přehled přenášených jednotek dat	67
9.2.2	Přehled způsobů přenosu dat.....	67
9.3	ROZDÍL LABORATORNÍ A INTERNETOVÉ VERZE	67
9.3.1	Stahování dat	68
9.3.2	Odesílání dat.....	69
10	TESTOVACÍ APLIKACE	70
10.1	SERVEROVÁ ČÁST	70
10.1.1	WebAPI.....	70
10.1.2	WCF	72
10.2	KLIENSKÁ ČÁST	74
10.3	ZMĚNA U VEŘEJNÉHO TESTOVÁNÍ.....	77
10.3.1	Serverová část	77
10.3.2	Klientská část	78
11	NAMĚŘENÉ HODNOTY	79
11.1	LABORATORNÍ MĚŘENÍ.....	79
11.1.1	Výsledky testu přenášení serializovaného objektu	80
11.1.2	Výsledky testu stahování binárních dat.....	82
11.1.3	Výsledky testu nahrávání binárních dat	87
11.1.4	Zaznamenané průběhy přenášení dat v čase.....	90
11.2	VEŘEJNÉ MĚŘENÍ.....	92
11.2.1	Výsledky testu stahování binárních dat.....	93
11.2.2	Výsledky testu nahrávání binárních dat	95

12	URČENÍ REŽIE Z PŘENÁŠENÝCH DAT	96
12.1	SROVNÁNÍ VELIKOSTI REŽIE JEDNOTLIVÝCH ZPŮSOBŮ PŘENOSU	96
12.2	SROVNÁNÍ VELIKOSTI PŘENESENÝCH DAT PŘI VOLÁNÍ FUNKCE	99
12.2.1	Rozdíl mezi vyhodnocením v kap. 12.1 a 12.2.	101
13	DOPORUČENÍ ZPŮSOBU PŘENOSU DAT	102
13.1	KOMUNIKACE MEZI ZAŘÍZENÍMI PODPORUJÍCÍMI .NET	102
13.2	KOMUNIKACE SE ZAŘÍZENÍM NEPODPORUJÍCÍ .NET	102
14	PRAKTICKÝ PŘÍNOS PRÁCE	104
	ZÁVĚR	106
	ZÁVĚR V ANGLIČTINĚ	108
	SEZNAM POUŽITÉ LITERATURY	110
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	113
	SEZNAM OBRÁZKŮ	116
	SEZNAM GRAFŮ	117
	SEZNAM TABULEK.....	118
	SEZNAM PŘÍLOH.....	119

ÚVOD

S vývojem informačních technologií se stále zvyšují nároky na prováděné operace a množství uchovávaných dat. Současně s těmito nároky musí růst výpočetní výkon systémů, které dané operace a ukládání dat provádějí. Při využití současných výrobních technologií není možno nadále výrazně zvyšovat pracovní frekvenci jádra procesoru a je nutno hledat jiné způsoby, které vedou k nárůstu výpočetního výkonu. Začíná se používat paralelizace na úrovni jednoho výpočetního uzlu, jenž může obsahovat více jádrový procesor, více jedno jádrových procesorů, nebo kombinace obou předchozích případů.

Při potřebě obrovského paralelního výkonu lze jít ještě dál. Celý systém je možno postavit z mnoha výpočetních uzlů, které mohou být umístěny v jedné místnosti, ale také rozmístěny po celém světě. Na všech uzlech může běžet z pohledu vnějšího světa jedna aplikace a každý z uzlů obsahuje část funkčnosti této aplikace. Uzel může sloužit jako jednotka zpřístupňující aplikaci vnějšímu světu, vnitřní výpočetní jednotka, mezilehlá komunikační jednotka nebo jednotka pro úschovu dat. V systému pak mohou existovat obrovské výpočetní a datové clustery, které dokáží řešit komplexní úlohy a obsluhovat současně mnoho vnějších klientů.

Výpočetní uzly spolu potřebují nějak komunikovat. Nějakým způsobem si musí vzájemně předávat data a vzájemně mezi sebe rozdělovat úkoly. Za předpokladu unifikované komunikace nezáleží na samotné hardwarové a softwarové architektuře jednotlivých uzlů. Tato práce se zabývá způsoby, jakými mohou uzly mezi sebou komunikovat a formátem předávaných dat. Cílem je nalézt nejvhodnější formát v závislosti na požadované funkcionalitě systému a rychlosti přenosu dat tak, aby byly plně uspokojeny požadavky na systém a komunikace probíhala co nejefektivněji.

Z důvodu spolupráce s firmou Business Logic s.r.o. je práce zaměřena na komunikační technologie v prostředí .NET, jmenovitě WebAPI a WCF. Teoretická část se věnuje nezbytné teorii pro část praktickou. Praktická část se zabývá testováním vybraných způsobů komunikace technologie WebAPI a WCF z hlediska délky trvání volání funkce, při kterém se přenáší různé typy a množství dat, a velikosti režie při přenosu. Testování probíhalo v laboratorních i veřejných podmínkách. Naměřená data jsou statisticky zpracována a na základě výsledků jsou dle klíčových požadavků doporučeny jednotlivé technologie, protokoly a jejich nastavení.

I. TEORETICKÁ ČÁST

1 ÚVOD DO DISTRIBUOVANÝCH SYSTÉMŮ

1.1 Co je to distribuovaný systém

Pro potřeby této práce je distribuovaný systém definován jako soustava výpočetních / datových uzlů, které spolupracují na řešení úkolů. Tyto uzly mohou být umístěny v jedné lokaci, nebo mohou být rozmístěny po celém světě. Všechny uzly může spravovat jedna organizace, nebo mohou mít jednotlivé části systému rozdílné správce.

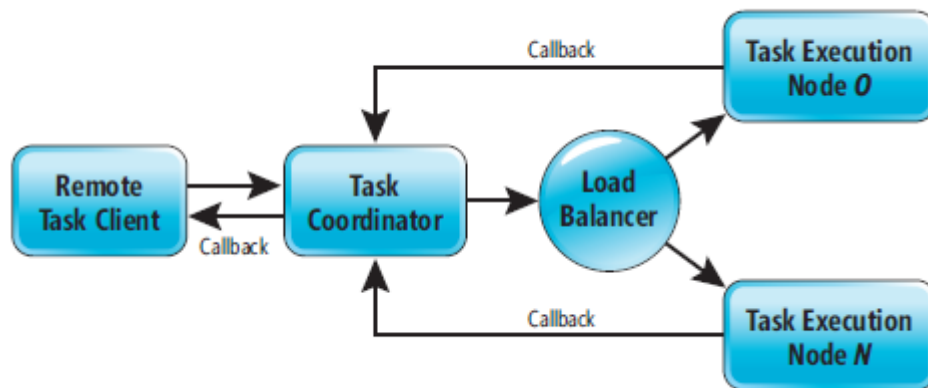
Uzly nemusí běžet na jedné hardwarové architektuře a mohou mít rozdílné operační systémy, aby se uzly spolu dokázaly domluvit, používají standardizované komunikační protokoly. Pokud je celý systém postaven na souhlasné softwarové architektuře, je možno s výhodou využít proprietárních protokolů pro přenos dat.

1.2 K čemu je distribuovaný systém dobrý

1.2.1 Sdílení výkonu

Systém se navenek může tvářit jako jeden celek poskytovat pro komunikaci s ním jedno unifikované rozhraní. Přes toto rozhraní se systému zadávají úkoly, které je nutno zpracovat. Systém jednotlivé úkoly rozděluje na jednotlivé výpočetní uzly, které provedou výpočty a vrátí výsledky.

Systém může přijaté úkoly vkládat do fronty a na základě různých kritérií, jako jsou například typ úkolu, priorita úkolu, předpokládaný čas dokončení úkolu a vytíženost jednotlivých výpočetních uzlů, tyto úkoly přiřazovat na jednotlivé výpočetní uzly. Při plánování jednotlivých úkolů je důležité, aby byly kritické úkoly splněny včas a aby nedošlo k přehlcení a pádu systému.



Obr. 1: Organizace distribuovaného systému pro sdílení výkonu [1, s. 20]

1.2.2 Distribuce dat

Data mohou být podle určitých pravidel rozložena v různých částech systému. Ve velkých datových centech se při zaplnění úložiště na jednom uzlu automaticky ukládají do úložiště jiného uzlu. Distribuce dat může být rovněž pojata z hlediska geografického, tak, aby byla data co nejbližší k uživateli, který s nimi pracuje.

Systém by měl splňovat ACID princip - Atomic, Consistent, Isolated, Durable [2].

- Atomické operace s daty, buď se operace dokončí celá, nebo neproběhne vůbec.
- Do transakce vstupují a z transakce vystupují konzistentní data.
- Mezivýsledky nejsou nikomu zpřístupněny.
- Po dokončení transakce jsou k dispozici výsledky.

Pro distribuované systémy platí tzv. CAP teorém Consistent, Available, Partition Resilient

- Consistent – data jsou shodná ve všech kopiích.
- Available – data jsou v systému aktivně ve více kopiích, pracuje se s více kopiemi současně (vyšší výkon a dostupnost).
- Resilience - odolnost proti výpadku replikace dat, která udržuje data shodná ve všech kopiích.

Podle CAP teorému mohou být spolehlivě splněny pouze 2 body z těchto 3.

Vysvětlení [2]:

1. Data jsou pouze v jedné kopii.
2. Z důvodu zvýšení dostupnosti a ochrany proti ztrátě dat se data duplikují do několika kopií a se všemi kopiemi se pracuje současně nezávisle na sobě.
3. Jednotlivé kopie se musí dostatečně rychle synchronizovat.
4. Pokud vypadne synchronizační spojení mezi kopiemi dat, mohou nastat dvě možnosti.
 - a) Systém pokračuje v činnosti, data nelze synchronizovat, ztrácí se konzistence.
 - b) Systém je zastaven, konzistence dat je zachována, ale systém není dostupný.

Z výše uvedeného vyplývá, že není možné vytvořit nekonečně škálovatelný systém, který by splňoval ACID princip.

Jelikož je princip ACID těžké dodržet, byl vytvořen princip BASE, který říká, že některé části systému mohou být nefunkční, ale systém musí stále zachovávat základní funkcionalitu, data nemusí být konzistentní okamžitě, ale konzistence musí být dosaženo později.

1.2.3 Rozdělení funkcionality na jednotlivé uzly

Distribuovaného systému lze s výhodou využít pro rozdělení funkcionality na jednotlivé uzly, kde každý z nich provádí pouze určitou specifickou činnost, která je součástí celku.

Dejme tomu, že systém plní komplexní proces zpracování dat. Určitá část zpracování může být provedena na jednom uzlu, z části zpracovaná data se předají dál dalšímu uzlu, který nad nimi provede další operaci a předá je dál. Data postupně projdou danou cestou v systému a na výstupu systému jsou již plně zpracovaná.

1.3 Distribuované aplikace

Pojem distribuovaná aplikace je pro potřeby této práce definován jako aplikace, jejíž celková funkcionalita je rozdělena na více fyzických uzlů. Základní popis tohoto řešení je uveden v kapitole 1.2.3. Distribuovaná aplikace se skládá z několika částí, kde každá část běží na jiném počítači. Určitá část může zpracovávat požadavky od klientů, jiná může provádět časově náročné výpočetní operace, datová část se může plně starat o ukládání a načítání dat z datového úložiště. Mezi jednotlivými částmi aplikace vzniká určité abstraktní komunikační rozhraní, pomocí něž jednotlivé části aplikace komunikují. Každá dílčí část se tedy může plně soustředit jen na svůj úkol a jediné, co potřebuje vědět je, jak komunikovat s jinými částmi.

Proto, aby mohla mezi jednotlivými částmi aplikace probíhat komunikace, musí každá část poskytovat navenek komunikační bod, ke kterému se mohou ostatní části aplikace v případě potřeby připojit. Skrze tento bod mohou dané části aplikace sdělit, jakou činnost má provést a současně ji předat a vyzvednout si od ní data, která se této činnosti týkají.

1.3.1 Komunikace v distribuovaných aplikacích

Existují dva základní způsoby komunikace, které mohou být použity. Rozdíl mezi nimi spočívá v tom, jakým způsobem jsou přenášeny informace o tom, co se má provést a způsob, jak se předávají data.

Prvním druhem je SOAP (Simple Object Access Protocol, jednoduchý protokol pro přístup k objektům), aplikace postavené na tomto druhu komunikace používají servisně orientovanou architekturu (Service Oriented Architecture, SOA). U tohoto způsobu komunikace se data zasílají vždy na stejný komunikační bod (stejnou adresu) a v těle zprávy je ve formátu XML (Extensible Markup Language, rozšiřitelný značkovací jazyk) uvedeno, jaká funkce se má na cílovém uzlu provést a jaké hodnoty se mají dosadit jako její parametry. Výhody tohoto řešení jsou vysoké a vyspělé možnosti zabezpečení a odolnosti proti chybám. Mezi nevýhody patří vysoká režie při přenosu a potřeba speciálního klienta, který daný protokol podporuje. Z tohoto důvodu se SOAP nejčastěji používá jako interní komunikace uvnitř distribuované aplikace [3].

Druhým druhem je REST (Representational State Transfer, reprezentační přenos stavu), aplikace postavené na tomto druhu komunikace používají zdrojově orientovanou

architekturu (Resource Oriented Architecture, ROA). Tento způsob komunikace používá stejně jako webové stránky protokol HTTP (Hyper Text Transfer Protocol, protokol pro přenos hyper textu). Funkce, která se má na cílovém uzlu provést, je získána kombinací části URL adresy (Uniform Resource Locator) s klíčovým slovem HTTP protokolu (GET/PUT/POST/DELETE), parametry této funkce jsou buď získány z URL, nebo z hlaviček HTTP protokolu. Výhody tohoto řešení jsou nízká komunikační režie a podpora téměř všude, kde je podporován protokol HTTP. Nevýhody řešení jsou nízké možnosti zabezpečení a velmi omezené možnosti ochrany proti chybám na úrovni samotné komunikace. Tento způsob je nejčastěji používán pro zpřístupnění distribuované aplikace okolnímu světu [3].

Nelze objektivně tvrdit, který přístup je absolutně lepší, protože každý je vhodný na jiný druh propojení. O čem ale lze diskutovat, je vhodnost použití jednotlivých druhů komunikace v závislosti na požadované funkcionalitě a způsobu nasazení. Samotné způsoby komunikace a formát přenášených dat budou podrobněji probrány v následujících kapitolách.

2 BĚHOVÉ PROSTŘEDÍ .NET

.NET platforma od Microsoftu (a programovací jazyk C#) byly formálně uvedeny v roce 2002. Tato platforma umožňuje stavět systémy na operačních systémech rodiny Windows, existují neoficiální implementace i pro Linux a Mac OS X. [4, s. 3]

Vlastnosti platformy .NET:

- Interoperabilita s existujícím kódem (COM).
- Podpora více programovacích jazyků (C#, Visual Basic, F# a další).
- Běhové prostředí sdílené všemi .NET jazyky (CLR).
- Podpora dědičnosti mezi více různými jazyky.
- Obsáhlá základová knihovna funkcí (BCL).
- Interpretovaný kód a automatická správa paměti.

2.1 Součásti .NET

Z programátorského hlediska může být .NET pochopeno jako běhové prostředí a obsáhlá základová knihovna. Běhová vrstva se nazývá Common Language Runtime (CLR, běžné jazykové běhové prostředí) a její primární účel je lokalizovat, načíst, a spravovat .NET objekty. CLR se také stará o spoustu nízko-úrovňových detailů, jako jsou správa paměti, hostování aplikace, koordinování systémových vláken a provádění bezpečnostních kontrol.

Další součástí je Common Type System (CTS, běžný typový systém). Specifikace CTS plně popisuje všechny možné datové typy a všechny programovací konstrukty, které jsou podporovány běhovým prostředím, specifikuje, jak tyto entity mohou vzájemně interagovat a detaily, jak jsou tyto entity reprezentovány v .NET metadatach.

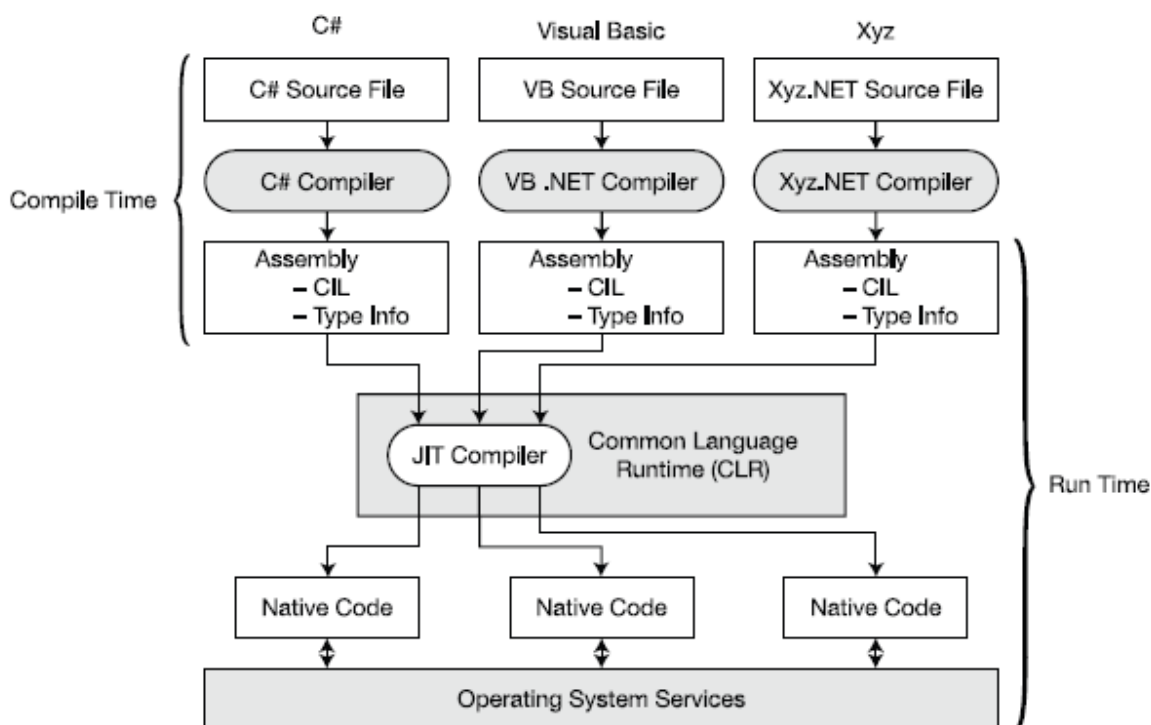
Ne všechny .NET jazyky musí podporovat vše, co je definované v CTS. Z tohoto důvodu byla vytvořena Common Language Specification (CLS, běžná jazyková specifikace), která popisuje podmnožinu běžných typů a programových konstruktů, které jsou podporovány úplně všemi .NET jazyky. [4, s. 4]

Mezi další součásti patří Base Class Library (BCL, základová knihovna tříd), která je k dispozici všem programovacím jazykům. Tato knihovna zaobaluje primitiva jako vlákna,

souborový vstup / výstup, systém vykreslování grafiky, interakce s externími zařízeními a podporu služeb, které jsou vyžadovány běžnými aplikacemi. [4, s. 5]

2.2 Kompilace

Aplikace napsané v prostředí .NET nejsou přímo kompilovány do binárního kódu, místo toho jsou pouze přeloženy do jazyka CIL¹ (Common Intermediate Language, běžný prostřední jazyk). Teprve běhové prostředí za chodu překládá tento jazyk do binárního kódu dle zvoleného operačního systému a hardwarové platformy. Tento bod je ilustrován obrázkem



Obr. 2: .NET – průběh kompilace [5]

¹ CIL je někdy též nazýván MSIL (Microsoft Intermediate Language, prostřední jazyk Microsoftu) nebo též jen IL (Intermediate Language, prostřední jazyk).

2.3 Technologie postavené na prostředí .NET

2.3.1 Windows Forms

Umožňuje vytvářet formulářové okenní desktopové aplikace. Jedná se o starší technologii, která zaobaluje standardní okna a ovládací prvky Win32 API a umožňuje je používat v prostředí .NET.

2.3.2 WPF

Windows Platform Foundation je nová technologie na vytváření desktopových okenních aplikací. Není již tak závislá na původním Win32 API jako Windows Forms. Grafické rozhraní se vytváří pomocí značkovacího jazyka XAML (Extensible Application Markup Language, rozšiřitelný aplikační značkovací jazyk).

2.3.3 ASP.NET

Active Server Pages .NET slouží k dynamickému zpracování webového obsahu na straně serveru. Dělí se na další pod technologie – Web Forms a MVC. Web Forms jsou obdoba technologie WinForms (webové stránky), MVC je implementace paradigmatu Model-View-Controller, který umožňuje oddělení dat, zobrazení a aplikační logiky a snadno prezentovat tatáž data různými způsoby.

2.3.4 WCF

Windows Communication Foundation umožňuje počítačům vzájemně komunikovat pomocí vzdáleného volání metod. Poskytuje široké možnosti konfigurace v oblasti přenosu zpráv a umožňuje přenos dat přes protokoly HTTP, TCP, MSMQ, Named Pipes a další.

2.3.5 LINQ

Language Integrated Query je technologie, která umožňuje s kolekcemi objektů pracovat podobně, jako s tabulkami v databázi.

2.3.6 WF

Windows Workflow Foundation poskytuje nástroje umožňující modelovat proces jako sérii kroků, kde každý krok je určitá aktivita, která se má provést. Tok programu je řízen

pomocí podmínek, které jsou vyhodnocovány v rozhodovacích blocích. Návrh aplikační logiky se velmi podobá aktivitnímu diagramu používanému při návrhu chování aplikace v BPMN (Business Process Model and Notation, zápis a model business procesu).

3 DATOVÉ FORMÁTY POUŽÍVANÉ PŘI PŘENOSU

Z důvodu kompatibility přenosu dat mezi různými systémy byly časem standardizovány dva formáty – XML a JSON.

3.1 XML

XML (Extensible Markup Language, rozšiřitelný značkovací jazyk) se vyvinul z jazyka SGML (Standard Generalized Markup Language, standardní generalizovaný značkovací jazyk), který zase vznikl z jazyka GML (IBM Generalized Markup Language, zobecněný značkovací jazyk IBM). Jazyk GML vznikl již v 60. letech minulého století a původně sloužil pro formátování textu při tisku. Do původního textu se doplnily řídicí informace, které tiskovému stroji říkaly, která část textu je nadpis, kde začíná další odstavec, co má být tučně a podobně. [6]

Jazyk XML byl doporučen W3C dne 10. února 1998 jako standardní formát pro výměnu informací v prostředí Internetu [7].

Cílem bylo navrhnout jazyk, který [7]:

- By měl být přímo použitelný na Internetu.
- By měl podporovat širokou škálu aplikací.
- By měl být kompatibilní s jazykem SGML.
- Mělo by být snadné psát programy, které zpracovávají XML dokumenty.
- Počet volitelných vlastností v XML bude držen na absolutním minimu, ideálně žádné.
- XML dokumenty by měly být čitelné pro člověka a dostatečně jasné.
- Návrh XML by měl být připraven rychle.
- Návrh XML by měl být formální a konzistentní.
- XML dokumenty by mělo jít snadno vytvořit.
- Stručnost značkování má minimální význam.

Poslední revizí jazyka XML verze 1.0 je Fifth Edition (5. vydání) z data 26. listopadu 2008 [8]. Existuje rovněž verze 1.1, která přidává podporu znaků dalších jazyků, ale běžně se nepoužívá.

3.1.1 Základní formát jazyka XML

Jazyk XML popisuje dokument pomocí tzv. tagů, tag slouží k označení dat, které uzavírá. Tag určuje, jak se s daty bude dále pracovat a co jednotlivá data znamenají. Každý tag začíná znakem < a končí znakem >. Aby byl dokument tzv. well formed, musí být každý tag uzavřen pomocí znaku /. Možné řešení tedy jsou <tag></tag> a <tag />. Mezi tagy se mohou nacházet data, tagy je možné do sebe vnořovat a vytvářet stromové struktury.

Základní struktura dokumentu XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<zakaznici>
  <zakaznik>
    <jmeno>Josef</jmeno>
    <prijmeni>Novák</prijmeni>
  </zakaznik>
  <zakaznik>
    <jmeno>Eva</jmeno>
    <prijmeni>Nováková</prijmeni>
  </zakaznik>
</zakaznici>
```

Na příkladu lze vidět, že dokument má jeden kořenový element nazvaný „zakaznici“, který má dále potomky v podobě elementů „zakaznik“, každý „zakaznik“ je člověk, který má nějaké vlastnosti, zde „jmeno“ a „prijmeni“.

Každý tag může mít také atributy, které mohou poskytnout další informace týkající se dat v tagu.

```
<zakaznik id="2096">
  <jmeno>Josef</jmeno>
  <prijmeni>Novák</prijmeni>
</zakaznik>
```

Na příkladu výše je k tagu „zakaznik“ přidán atribut „id“. Do určité míry je možné z hlediska ukládání dat do XML zaměňovat atributy s hodnotou elementů, kteří jsou potomci. Je tedy možné napsat:

```
<zakaznik id="2096" jmeno="Josef" prijmeni="Novák" />
```

```
ale i <zakaznik>
      <id>2096</id>
      <jmeno>Josef</jmeno>
      <prijmeni>Novák</prijmeni>
</zakaznik>
```

Oba zápisy obsahují stejná data, způsob zápisu je ale odlišný. To, který způsob zápisu použít závisí čistě na rozhodnutí člověka, který danou strukturu XML dat vymyslel, důležité ale je, aby byla na obou stranách komunikace podporována stejný formát zápisu.

3.1.2 Definice typu dokumentu

Aby byla struktura dokumentu shodná pro obě komunikující strany, byl vytvořen prostředek umožňující formální popis struktury dokumentu, tzv. DTD (Document Type Definition, definice typu dokumentu). Pokud struktura XML dokumentu odpovídá deklarovanému DTD, je dokument považován za validní dle dané DTD specifikace.

DTD určuje, jaký typ dat může obsahovat daný element, které elementy jsou v dokumentu povinné, jaké potomky může obsahovat daný element, v jakém pořadí musí potomci daného elementu být, jaké atributy může obsahovat daný element, jaká je výchozí hodnota atributu pro daný element, pokud není atribut u daného elementu uveden a jaké atributy musí daný element obsahovat, aby byl dokument považován za validní.

Například [6]:

```
<?xml version="1.0" encoding="UTF-8">
<!ELEMENT message-break EMPTY>
<!ELEMENT data (#PCDATA)>
<!ELEMENT message (data | message-break)*>
```

říká, že element „message-break“ musí být vždy prázdný, element „data“ obsahuje parsovaná data (Parsed Character Data, tato sekvence nemůže obsahovat vyhrazené znaky XML, jako je například <> & apod., tyto znaky se nahrazují ekvivalenty < > &)

a element `message` může obsahovat buď podelement „`data`“ nebo podelement „`message-break`“, * znamená, že element „`message`“ může obsahovat víc podelementů.

XML dokument, který by byl validní dle daného DTD by mohl vypadat třeba takto:

```
<message>
  <data>Toto jsou nějaká data 1</data>
  <data>Toto jsou nějaká data 2</data>
  <message-break/>
  <data>Toto jsou nějaká data 3</data>
</message>
```

V případě, že je nutno v hodnotě tagu nebo v jeho attributech používat zakázané XML znaky, je nutno daný textový řetězec obalit znaky `<![CDATA[text]]>`. CDATA znamená Character Data (znaková data), obsah toho prvku není parsován a není při validaci DTD kontrolován.

3.1.3 Jmenné prostory

Aby bylo možno odlišit různé implementace elementů se stejným názvem, byly zavedeny jmenné prostory, pomocí nichž lze dané elementy odlišit.

Příklad jmenných prostorů uveden níže [9]:

```
<root>
  <h:table xmlns:h="http://www.w3.org/TR/html4/">
    <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
    </h:tr>
  </h:table>
  <f:table xmlns:f="http://www.w3schools.com/furniture">
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
  </f:table>
</root>
```

V dokumentu se vyskytuje dvakrát element „`table`“ a pokaždé má jiné pod-elementy. Aby bylo možné tyto dva druhy elementu „`table`“ při parsování odlišit, je pro každý element „`table`“ definován pomocí atributu „`xmlns`“ (XML namespace) jmenný prostor ve formátu

URI (Uniform Resource Identifier, unikátní identifikátor zdroje). Jmennému prostoru je přiřazen zástupný symbol, který dále slouží jako prefix pro elementy, které přísluší do daného jmenného prostoru.

Jmenný prostor nemusí být vyjádřen pouze ve formátu URI, může se jednat o jakýkoliv souvislý text. Formát URI je zde volen z důvodu zamezení kolizí jmen jmenného prostoru.

3.1.4 XML schéma

XML schéma, často nazývané XSD (XML Schema Document, dokument XML schématu) je nástupcem DTD (viz. kap. 3.1.2). Oproti DTD přidává podporu jmenných prostorů, je celé zapsáno v XML a umožňuje definovat datový typ dat a parametrů.

Například v souboru „poznamka.xsd“ je schéma definováno takto [10]:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

  <xs:element name="poznamka">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="komu" type="xs:string"/>
        <xs:element name="od" type="xs:string"/>
        <xs:element name="nadpis" type="xs:string"/>
        <xs:element name="telo" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

xmlns:xs="http://www.w3.org/2001/XMLSchema" indikuje, že elementy a datové typy pocházejí ze jmenného prostoru „http://www.w3.org/2001/XMLSchema“ a používají prefix „xs“ [10]

targetNamespace="http://www.w3schools.com" indikuje, že elementy definované tímto schématem pocházejí ze jmenného prostoru „http://www.w3schools.com“ [10]

xmlns=http://www.w3schools.com indikuje, že výchozí jmenný prostor je „http://www.w3schools.com“ [10]

elementFormDefault="qualified" indikuje, že všechny elementy použité XML instancí dokumentu, které byly deklarovány v tomto schématu, se kvalifikují pro jmenný prostor

v souboru „poznamka.xml“ potom může být použito takto [10]:

```
<?xml version="1.0"?>
<poznamka
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com poznamka.xsd">
    <komu>Honza</komu>
    <od>Pepa</od>
    <nadpis>Připomínka</nadpis>
    <telo>Nezapomeň na mě tento víkend!</telo>
</poznamka>
```

xmlns="http://www.w3schools.com" specifikuje výchozí jmenný prostor, říká validátoru, že všechny elementy použité v tomto dokumentu používají daný jmenný prostor [10]

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" specifikuje jmenný prostor instance XML schématu [10]

xsi:schemaLocation="http://www.w3schools.com poznamka.xsd" první hodnota udává, který jmenný prostor se má použít, druhá umístění XML schématu pro daný jmenný prostor [10]

3.2 JSON

JSON (JavaScript Object Notation, zápis objektů JavaScriptu) je „lehký“ textový formát pro výměnu dat. Byl navržen s ohledem na snadnou čitelnost a zápis jak pro člověka, tak pro počítač. Formát je založen na podmnožině programovacího jazyka JavaScript, standardu ECMA-262 3rd Edition (prosinec 1999). [11]

3.2.1 Formát jazyka JSON

Jazyk JSON umožňuje ukládat tři základní typy dat – objekty, pole a hodnoty. Objekt se skládá z jedné nebo více { klíč : hodnota } položek, které jsou odděleny čárkou. Pole potom může obsahovat více hodnot, objektů nebo polí [hodnota1, hodnota2].

JSON podporuje tyto typy hodnot [12]:

- Číslo (integer nebo s plovoucí řádovou čárkou).
- Textový řetězec (v uvozovkách "").
- Boolean (true nebo false).
- Pole (v hranatých závorkách []).
- Objekt (v složených závorkách {}).
- null

Objekt:

```
{ "jmeno": "Jan" , "prijmeni": "Novotný" }
```

Objekt, který obsahuje pod klíčem „zamestnanci“ pole objektů, pod klíčem „datum“ textový řetězec a pod klíčem „id“ číselnou hodnotu:

```
{  
  "zamestnanci": [  
    { "jmeno": "Jan" , "prijmeni": "Novotný" },  
    { "jmeno": "Eva" , "prijmeni": "Nováková" },  
    { "jmeno": "Peter" , "prijmeni": "Jones" }  
  ],  
  "datum": "10.3.2013",  
  "id": 2512  
}
```

3.3 Porovnání XML a JSON

Výhodou XML jsou větší možnosti při reprezentaci dat - všechny jednotlivé elementy jsou pojmenovány, podpora atributů, jmenných prostorů a validace pomocí definice typu dokumentu nebo schémat. Nevýhodou XML je vysoká režie při přenosu.

Naproti tomu JSON poskytuje jen základní prostředek pro přenos objektů, kde jsou některá data předávána pouze pozičně a nezachovávají se ani všechna jména identifikující jednotlivé položky. Neposkytuje ani atributy, ani jmenné prostory, ani validaci. Výhodou formátu JSON je nízká režie při přenosu dat.

4 SÍŤOVÁ KOMUNIKACE NA NIŽŠÍCH VRSTVÁCH

V této kapitole budou zběžně probrány technologie, na které bude později odkazováno v praktické části práce při vyhodnocování výkonnosti různých typů komunikace na vyšších vrstvách. Kapitola se snaží prakticky ukázat, co se s daty na jednotlivých nižších vrstvách děje. Cílem zde není popsat všechny existující technologie a jejich varianty, ale pouze poskytnout teoretický základ pro vybrané konkrétní technologie, které budou sloužit jako „podstava“ pro vyhodnocované způsoby komunikace. Pokud nebude uvedeno jinak, používají všechny způsoby komunikace, které jsou uvedeny dále, na nižších vrstvách modelu ISO/OSI tyto protokoly:

1. Fyzické propojení kabelem UTP (CAT 5e).
2. Ethernet (1 Gbit/s, 802.3ab).
3. Internet Protokol verze 4 (IP v4).
4. Transmission Control Protocol (TCP).

Pro pozdější úvahy je nutná především základní znalost principu průchodu dat jednotlivými protokoly a režie, které tyto protokoly způsobují.

4.1 TCP (Transmission Control Protocol)

Data z vyšší vrstvy jsou nejprve předána protokolu TCP (Transmission Control Protocol, protokol řídicí přenos). TCP funguje na principu virtuálního spoje. Obě strany se nejdříve spolu domluví na tom, že spolu budou komunikovat a vytvoří tzv. TCP spojení. TCP spojení je identifikováno IP adresou, což je jedinečný identifikátor komunikačního uzlu v Internetu, a portem, který říká, kterého procesu² se daná komunikace týká. Spojení existuje mezi dvěma procesy buď na tomtéž jednom, nebo dvěma propojenými počítači. Data jsou v rámci daného spojení nejdříve doručeny pomocí IP adresy danému počítači a dále v rámci počítače danému procesu na základě čísla portu.

² Proces je, pro jednoduchou představu, spuštěný počítačový program, který buď s uživatelem přímo interaktivně komunikuje, nebo běží v rámci operačního systému na pozadí.

Z hlediska programu se TCP spojení chová jako bytově orientovaný datový proud, do kterého se data na jednom konci zapisují a z druhého konce čtou. Komunikace je plně duplexní, což znamená, že může probíhat souběžně v obou směrech. Z hlediska člověka si lze TCP spojení představit jako rouru, která vede mezi oběma konci spojení a cokoliv, co se do roury naleje z jedné strany, vyteče přesně tak, jak bylo posláno, z druhé strany. TCP garantuje doručení dat přesně tak, jak byly odeslána a sám na pozadí řeší doručování dat ve správném pořadí, znovu zaslání po cestě ztracených dat a řízení rychlosti toku dat tak, aby nedošlo k přetížení a následném kolapsu sítě.

4.1.1 Funkce protokolu TCP

Při zahájení komunikace se nejdříve pomocí tří kroků vytvoří spojení. Strana A vytvoří libovolné sekvenční číslo $S1$ a pošle ho druhé straně, se kterou chce komunikovat. Strana B toto číslo zvýší o 1 a pošle ho zpět jako potvrzující číslo ($A1=S1+1$), současně vytvoří náhodné sekvenční číslo $S2$ pošle jej straně A. Strana A následně sekvenční číslo $S2$ zvýší o 1 a odešle straně B jako $A2=(S2+1)$. Takto jsou obě strany domluveny na komunikaci.

Při komunikaci jsou data z vyšších vrstev přebírána v podobě datového proudu a rozdělována na dílčí části (tzv. segmenty), kde každá část obsahuje své „sekvenční číslo“. Sekvenční číslo udává číslo bytu (v rámci streamu), kterým má následující segment začínat. Tyto jednotlivé části (segmenty) jsou dále nezávisle na sobě předány nižším vrstvám a přeneseny na „druhou stranu“, kde jsou zase sestaveny do datového proudu a poskytnuty aplikaci.

Doručení segmentu dat je potvrzeno žádostí o nový segment dat s daným počátkem v rámci datového proudu a požadovanou délkou. Požadovaná délka je dána tzv. oknem.

Pokud je strana B ochotna přijímat data, pošle straně A zprávu o tom, že je ochotna přijmout data a kolik dat je schopna zpracovat. Pro jednoduchou ilustraci si je možno představit, že v prvním kroku chce stanice B od stanice A data od bytu 1 a že je schopna zpracovat 300B, nastaví tedy číslo $A1$ na 1 a okno na 300 a pošle toto stanici A. Stanice A z datového proudu vezme 300 bytů od pozice 1. bytu, tj. byty 1 až 300 nastaví číslo $S1$ na hodnotu 1, jelikož od stanice B momentálně žádná data nechce, nastaví $A2$ na 0 (dosud nepřijala žádná data) a okno na 0. Stanice B přijme data a nastaví číslo $A1$ na číslo následujícího bytu, který očekává, tedy 301 a okno na ponechá na 300. Číslem $A1=301$ potvrdila stanici A přijetí předchozích 300B a říká, že chce přijmout 300 bytů od bytu 301.

Číslo $A2=0$ poslané stanicí A říká stanici B, že stanice A je ochotna přijmout data od bytu 0 a že je ochotna přijmout 0 bytů, což znamená, že momentálně stanici A žádná data posílat nebude.

Daný příklad je pouze ilustrační. Reálně hodnoty $S1$, $A1$, $S2$, $A2$ nezačínají od 0, ale jsou náhodně generované při sestavování spojení. Důležitý je fakt, že stanice druhé sděluje, od jaké pozice chce momentálně přijmout data a kolik dat chce momentálně přijmout. Uvedením pozice rovněž potvrzuje správné přijetí předchozích dat. V případě, že se nějaká data při přenosu ztratí, tj. stanice B chce data od pozice 1 do pozice 300, ale přijdou ji jen data od pozice 1 do pozice 200, stanice B potvrdí stanici A správné přijetí pouze 200B dat tak, že v následujícím kole bude požadovat data od pozice 201 (tím dojde znovu k přenosu dat, která se ztratila/poškodila).

Mimo tyto jednoduché postupy potvrzování dat je protokol TCP schopen data potvrzovat také selektivně, tj. potvrdit přijetí bytů od určitého čísla S do určitého čísla S . Komplexnost protokolu TCP je daleko za rozsahem této práce.

Ukončení spojení probíhá pomocí 4 kroků (které mohou být spojeny do tří). Každá ze stran pošle druhé ukončovací FIN packet a druhá jeho přijetí potvrdí. Potvrzení přijatého FIN druhé strany a vyslání vlastního FIN může být spojeno do jednoho kroku.

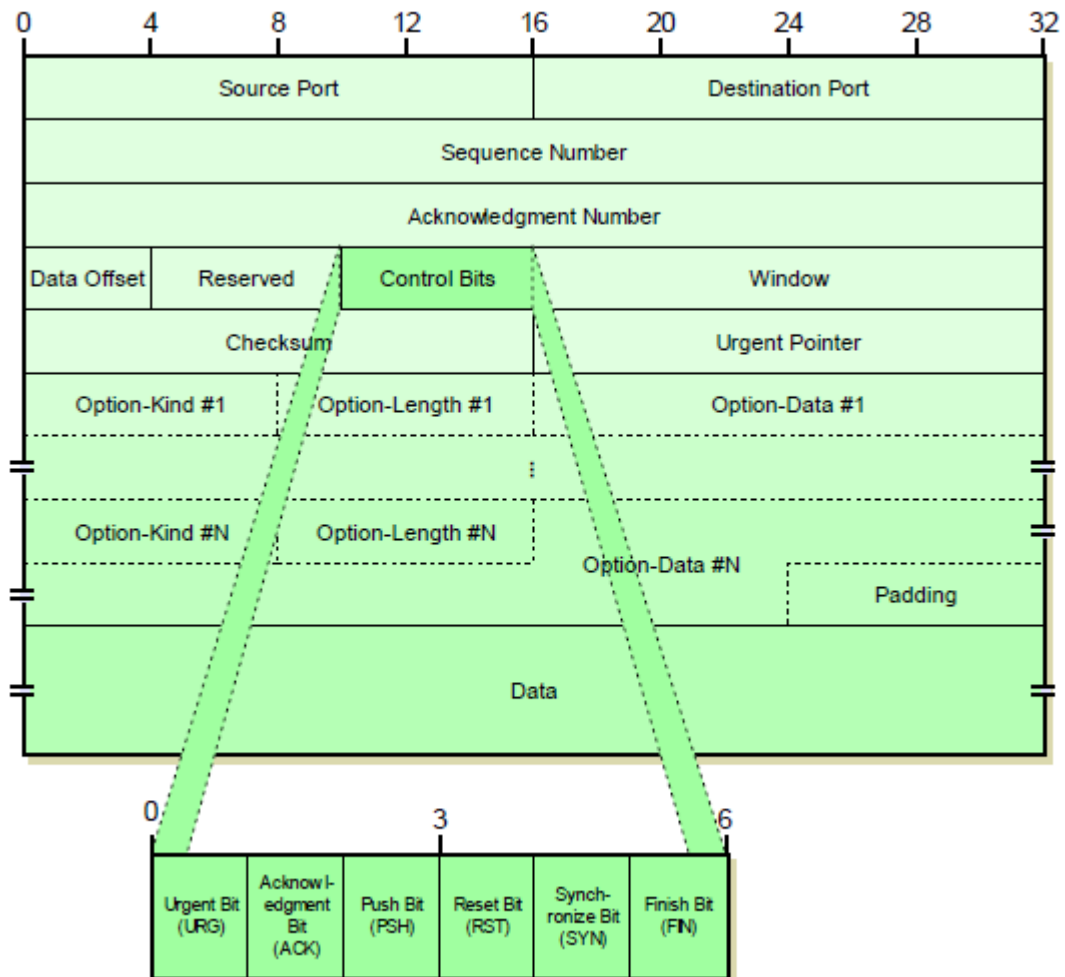
4.1.2 Režie protokolu TCP

Režie protokolu TCP je dána hlavně jeho hlavičkou, která je přidávána ke každému dílčímu požadavku na data. Právě zde jsou obsaženy čísla $S1$, $A2$ pro jeden směr a $S2$, $A1$ pro druhý směr komunikace. Velikost hlavičky je minimálně 20B a maximálně 60B v závislosti na rozšiřujících parametrech [13].

Hlavička obsahuje [13]:

- Zdrojový port (Source Port) identifikující odesílající proces.
- Cílový port (Destination Port) identifikující cílový proces.
- Sekvenční číslo (Sequence Number) udávající pozici zprávy v rámci datového proudu (zde číslo S).
- Potvrzující číslo (Acknowledgement Number) udávající pozici bytu, kterým má začínat následující zpráva v opačném směru (zde číslo A).

- Okno (Window) které říká, kolik bytů má obsahovat následující zpráva v opačném směru.
- Další informace a příznaky nutné k navázání, řízení a zrušení spojení (offset dat, řídicí bity, kontrolní součet).



Obr. 3: TCP Segment [13]

4.2 IP (Internet Protocol)

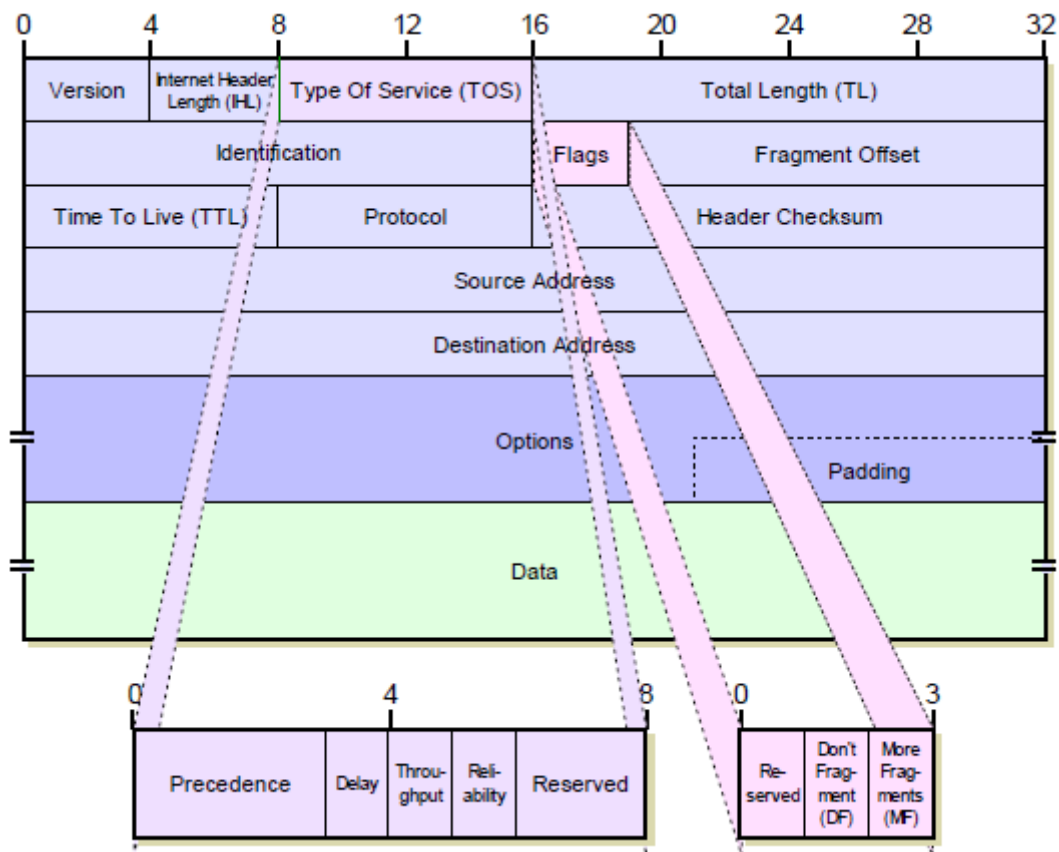
Segmenty, které protokol TCP vytvoří (případně datagramy, které UDP vytvoří), jsou předány protokolu IP (Internet Protocol, protokol mezi sítě), který ke každému segmentu (TCP hlavička + data) přidá další hlavičku. Pomocí této hlavičky je balíček dat směrován na routerech, které propojují jednotlivé sítě a umožňují přenášet data napříč Internetem. Hlavička má v případě protokolu verze 4 velikost 20B [13] (bez volitelných nastavení) a balíček dat v tomto stádiu zpracování se nazývá packet. Protokol verze 4, použitý zde

k uskutečnění záměru práce, používá 32 bitovou IP adresu ve tvaru xxx.xxx.xxx.xxx, kde xxx je číslo v rozmezí 0-255. Existují různé třídy IP adres, z hlediska této práce je pouze podstatné to, že dva body jsou v rámci TCP spojení identifikovány IP adresou a režie, kterou protokol IP při komunikaci přidá. IP adresa může být zastoupena doménovým jménem ve tvaru *treti_rad.druhy_rad.prvni_rad*. Příklad doménového jména je například *www.businesslogic.cz*. Pokud je některá z komunikujících stran zastoupena doménovým jménem. Je toto jméno před začátkem komunikace pomocí služby DNS přeloženo na IP adresu a komunikace následně navázána s touto IP adresou.

4.2.1 Režie protokolu IP

Režie protokolu IP je dána hlavičkou. Hlavička protokolu IP se skládá z [13]:

- Verze (Version) udávající verzi protokolu, pro verzi 4 obsahuje hodnotu 4.
- Velikost hlavičky (IHL) specifikuje velikost hlavičky v násobku 4 bytů.
- Typ služby (TOS) slouží k odlišení packetu v rámci služeb QoS.
- Celková délka datagramu v bytech (TL).
- Doba životnosti packetu (TTL) udává, po kolika směrovacích krocích bude packet zahozen.
- Protokol (Protocol) udává, kterému protokolu na vyšší vrstvě budou doručena data po zpracování protokolem IP předána, pro protokol TCP je uvedena hodnota 6.
- Zdrojová adresa (Source Address) specifikuje odesilatele packetu.
- Cílová adresa (Destination Address) specifikuje příjemce packetu.
- Další položky (kontrolní součet, příznaky, volitelné nastavení, offset rámce při fragmentaci).



Obr. 4: IP Packet [13]

4.3 Ethernet (IEEE 802.3)

Výstup protokolu IP, packet je dále zpracován protokolem Ethernet, který před packet (IP hlavička + TCP hlavička + data) přidá Ethernet hlavičku a za packet kontrolní součet. Takto vytvořený datový balíček se nazývá rámeček. Protokol Ethernet slouží pro síťovou komunikaci v rámci jedné lokální sítě, počítače jsou zde adresované pomocí 48 bitové MAC adresy (Media Access Control, řízení přístupu k médiu). Zatím, co se IP adresa nastavuje ručně nebo ji přiřadí operační systém, MAC adresa je napevno přiřazena a vypálena v čipu síťového adaptéru výrobcem hardwaru a měla by být světově unikátní. U adaptérů lze tuto adresu změnit.

V případě lokální sítě, když uzel zná IP adresu cílového uzlu, ale ne jeho MAC adresu, se použije protokol ARP (Address Resolution Protocol), který zjistí, jakou MAC adresu má síťové rozhraní s danou IP adresou. MAC adresa se určuje pro každou Ethernet síť, kterou

rámec prochází zvlášť a v případě, že obě komunikující strany nejsou ve stejné síti, není odesílateli adresa příjemce známa.

4.3.1 Režie protokolu Ethernet 802.3ab

Režie protokolu Ethernet je dána velikostí hlavičky (22 B), kontrolního součtu (4 B) a mezeře mezi rámci (12 B).

Ethernet rámec musí mít minimální délku 72 B, případně se na tuto délku doplňuje. Maximální délka je 1526 B. Mezi rámci je vždy mezera o délce 12 B. [13]

Rámec se skládá z [14]:

- Preambule (Preamble) – střídají se 1 a 0 (7 bytů).
- Oddělovač začátku rámce (SFD), 10101011, 1 byte.
- Cílová MAC Adresa (DA), 6 bytů.
- Zdrojová MAC Adresa (SA), 6 bytů.
- Délka dat (LT).
- Přenášená data (Data), 46 – 1500 bytů.
- Kontrolní součet (FCS), 4 byty CRC.

Mezi rámci je mezi rámcová mezera (IG), 12 bytů.

Preamble	SFD	DA	SA	LT	Data	FCS	IG
----------	-----	----	----	----	------	-----	----

Obr. 5: Ethernet rámec

5 ZPŮSOBY KOMUNIKACE

Mezi dva základní způsoby komunikace v distribuovaných aplikacích patří REST (Representational State Transfer, reprezentativní přenos stavu) a SOAP (Simple Object Access Protocol, jednoduchý protokol pro přístup k objektům). V této kapitole jsou tyto způsoby zhruba popsány. Pro přenos obou způsobů je nejčastěji využíván protokol HTTP (HyperText Transfer Protocol, protokol pro přenos hyper textu). Zde je protokol HTTP vysvětlen v kontextu principu REST, specifické použití tohoto protokolu pro SOAP je uvedeno v kapitole věnující se SOAP.

5.1 REST

REST je sbírka principů, kterými by se měl řídit návrh systému, tak, aby byly splněny požadavky vysoké kompatibility a snadné škálovatelnosti. Byl navržen Royem Fieldingem v roce 2000 v rámci jeho disertační práce *Architectural Styles and the Design of Network-based Software Architectures*. Nejčastější použití principů REST je v kombinaci s protokolem HTTP (HyperText Transfer Protocol, protokol pro přenos Hyper Textu), ale použití HTTP není podmínkou pro návrh systému odpovídající principům REST. [15]

Princip REST vyžaduje, aby klient nemusel dopředu znát všechny možné zdroje serveru, ale aby mu stačila pouze znalost vstupního bodu systému. Pomocí tohoto bodu si klient stáhne možné odkazy na zdroje, na které může později přistoupit. Lze to přirovnat k webové stránce, po zadání hlavní URL³ (Uniform Resource Locator, uniformní lokátor zdroje) se zobrazí menu, které dále webem návštěvníka naviguje, návštěvník tedy nemusí předem znát URL všech stránek webu, stačí mu pouze znalost vstupního bodu na web. Toto umožňuje snadno měnit obsah serveru bez nutnosti měnit klienta, klient musí pouze znát, jak strukturu odkazů nabízených serverem interpretovat. [15]

Aplikace je na serveru modelována jako jednoduchý stavový automat, kde každý stav může provádět nějakou operaci nebo vracet data. Každý stav aplikace je reprezentován tzv. zdrojem, který je přístupný přes určitý URL. Klientská aplikace musí vědět, v jakém pořadí mezi těmito stavy přecházet. Mezi URL a samotný zdroj je vkládána abstraktní vrstva,

³ URL je nejčastěji vyjádřen v podobě webové adresy, např. <http://www.tkdata.eu/pay?choice=yes>.

kteřá umožňuje při změně obsahu na serveru zachovat navenek stejné URL adresy, ale interně požadavek přeměrovat na změněný resource. Například pokud je na serveru kolekce dat, je možné vytvořit URL, které vždy odkazuje na naposledy vloženou položku kolekce, přičemž odkaz pokaždé ukazuje aktuální fyzický zdroj.

Zdrojem může být objekt reprezentován XML/JSON zápisem, obrázek, webová stránka, akce na serveru apod. Se zdroji lze obvykle provádět čtyři základní operace a to načtení zdroje (stažení ze serveru), vytvoření zdroje (nahrání na server), úprava zdroje, smazání zdroje.

Požadavky, které si při návrhu architektury vymínuje REST, jsou dále rozděleny na client/server, bezstavovost – každý požadavek musí obsahovat vše potřebné k jeho vykonání, každý požadavek musí být explicitně označen jako cachovatelný/necachovatelný, schopnost klienta přijmout kód ze serveru a provést jej a vrstevnatost, která umožňuje skládat vrstvy poskytující služby na sebe za účelem zajištění variabilnosti.

Termín REST je často nesprávně používán pro systémy, které používají URL pro vzdálené volání procedur, tento způsob komunikace ovšem neodpovídá původnímu záměru architektury REST a neměl by být takto nazýván. [15]

5.2 Protokol HTTP – nejčastější implementace principu REST

Princip REST je nejčastěji používán s protokolem HTTP (HyperText Transfer Protocol, protokol pro přenos Hyper Textu). HTTP protokol je bezstavový protokol typu dotaz-odpověď, kdy klient provede dotaz v určitém formátu na server, server dotaz vyhodnotí a odpoví klientovi v určitém formátu. Pro přenos se používá protokol TCP na portu 80, v přenosu se posílá text.

5.2.1 Dotaz na server

Formát dotazu klienta na server je následující:

```
GET /list HTTP/1.1
Host: tkdata.eu
User-Agent: Opera/9.80 (Windows NT 5.1; U; cs)
Accept-Charset: UTF-8,*
Accept: application/json
```

GET serveru říká, že chce klient zdroj stáhnout, server odpoví reprezentací zdroje, další možnosti jsou *HEAD*, kdy server pouze vrátí informace o zdroji, *POST* kdy server přijme data zasláná klientem, *PUT* kdy server upraví již existující zdroj a *DELETE*, kdy server smaže daný zdroj. Existují i další metody – *TRACE*, *OPTIONS*, *CONNECT* a další rozšíření ve WebDAV, ale tyto přímo nijak s REST architekturou nesouvisí. [16]

/list serveru říká, že zdroj, se kterým se pracuje, je zastoupen URL */list*, v rámci serveru se může jednat o pevnou adresu zdroje nebo o adresu která může v různé okamžiky ukazovat na různé fyzické zdroje

HTTP/1.1 vyjednává se serverem spojení ve verzi 1.1, verze 1.1 umožňuje efektivněji pracovat ze zdroji na daném serveru, protože dokáže provést více dotazů a přijmout více odpovědí v rámci jednoho TCP spojení, verze 1.0 pro každý dotaz a odpověď otvírala nové TCP spojení

Host: tkdata.eu určuje server, používá se k rozlišení virtuálního serveru v případě, že na serveru běží víc současných aplikací

User-Agent: Opera/9.80 (Windows NT 5.1; U; cs), *Accept-Charset: UTF-8,** a *Accept: application/json* jsou doplňkové (nepovinné) informace ve formátu klíč: hodnota, v této oblasti HTTP hlavičky může klient specifikovat libovolné doplňující informace pro server, například svoji identifikaci, jím podporovanou znakovou sadu, požadovaný formát dat apod. Server tyto informace využije k reprezentaci zdroje v klientem požadovaném formátu. Kombinace klíč: hodnota mohou být libovolné řetězce, důležité je, aby je server pochopil. Pokud server doplňující informace nepochopí nebo požadovaný formát nepodporuje, vrátí data ve výchozí reprezentaci.

Pozn:

Metody při dotazu na server lze třídit na bezpečné – *GET*, *HEAD*, *OPTIONS*, *TRACE* – tyto metody nezpůsobí na serveru žádné změny, nebezpečné – *POST*, *PUT*, *DELETE* – tyto metody způsobí změny na serveru, Idempotentní – *PUT*, *DELETE* - více stejných požadavků bude mít stejný výsledek jako jeden požadavek, Neidempotentní – *POST* – provedení stejného požadavku vícekrát nebude mít stejný výsledek jako provedení požadavku pouze jedenkrát.

5.2.2 Odpověď serveru

Server na požadavek na zdroj odpoví:

```
HTTP/1.1 200 OK
Date: Tue, 19 Mar 2013 19:30:17 GMT
Server: Apache/2.2.24 (Unix) PHP/5.4.13
Content-Type: application/json; charset=utf-8
{
  "list": [
    { "name": "Jiří" , "link": "/app/0000255/" },
    { "name": "Eva" , "link": "/app/0000173/" },
    { "name": "Peter" , "link": "/app/0000997/" }
  ]
}
```

HTTP/1.1 sděluje, že server souhlasí s protokolem verze 1.1

200 udává, jak dotaz dopadl. *200* znamená, že je vše v pořádku a server vrací v odpovědi data. Hodnoty v rozmezí 100 – 199 jsou informačního charakteru, 200 – 299 oznamují, že vše proběhlo v pořádku (například 200 – byla vrácena data, 201 – byl vytvořen nový zdroj, 203 – dotaz byl akceptován ke zpracování, ale zpracování dosud nebylo dokončeno), 300 – 399 informují o tom, že byl zdroj přesunut (300 – vícenásobná volba, 301 – permanentně přesunuto), 400 – 499 udávají chyby na straně klienta (například 400 – dotaz nebyl serverem pochopen, 401 – klient se musí nejdříve autorizovat, 404 – zdroj nenalezen), 500 – 599 udávají chyby serveru (500 – server se dostal do stavu, kdy není schopen splnit dotaz, 503 – služba nedostupná). Klient na základě tohoto kódu podnikne příslušné akce. [17]

OK je pouze informačního charakteru a nemá pro klienta při zpracování odpovědi význam, text popisuje stavový kód uvedený výše

Date: Tue, 19 Mar 2013 19:30:17 GMT uvádí datum a čas odpovědi

Server: Apache/2.2.24 (Unix) PHP/5.4.13 uvádí jméno a verzi serveru

Content-Type: application/json; charset=utf-8 informuje klienta o formátu odpovědi

Dále následuje odpověď serveru ve formátu JSON.

5.2.3 HTTPS

Protokol HTTPS (S – Security, bezpečnost) poskytuje zabezpečený přenos, při kterém jsou data šifrována a zabezpečena proti podvrhnutí. V podstatě se jedná pouze o vytvoření TLS (Transport Layer Security, bezpečnost transportní vrstvy) spojení nad již existujícím TCP

spojením. Všechna data, která jsou pak v rámci TCP spojení předávána, jsou automaticky transparentně šifrována a dešifrována.

Po vyjednání (používá se asymetrická kryptografie ve spojení s certifikátem serveru, případně certifikátem klienta a několika krokový handshake, jehož výsledkem, pokud je úspěšný, je dohodnutý klíč pro symetrickou šifru, data jsou dále šifrována symetricky) TLS spojení přes existující TCP spojení jsou HTTP data při odesílání místo přímého předání do TCP tunelu předána TLS prvku, který data zašifruje a teprve potom pošle TCP tunelem. Na druhé straně si data z tunelu převezme rovněž TLS prvek, dešifruje data a teprve předá HTTP vrstvě.

Jelikož jsou všechna zasílaná data šifrována, nelze zjistit obsah ani dotazu, ani odpovědi, jediné, co lze zjistit existence TCP spojení mezi dvěma body. Protokol TLS ovšem přidá do přenosu další režii, která je ovšem zanedbatelná. [18]

5.3 SOAP

SOAP (Simple Object Access Protocol, protokol pro přístup k jednoduchým objektům) byl navržen pro firmu Microsoft v roce 1998 autory Davem Winerem, Donem Boxem, Bobem Atkinsonem a Mohsenem Al-Ghoseinem. Nynější verze protokolu je 1.2 ze dne 27. dubna 2007 [19]. Komunikace běžně probíhá zasíláním zpráv založených na XML (viz. kap. 3.1) přes protokol HTTP (metoda POST) nebo SMTP (Simple Mail Transfer Protocol, protokol pro jednoduchý přenos pošty). Pokud služba používá pro přenos protokol HTTP, nazývá se tzv. webová služba. SOAP popisuje základní strukturu zprávy, pravidla pro zpracovávání zpráv, způsob rozšíření zpráv a svázání s protokolem na nižší vrstvě.

Zpráva protokolu SOAP se skládá z prvku Envelope (obálka), který v sobě obsahuje prvky Header (hlavička) a Body (tělo). Hlavička slouží pro informace týkající se přenosu dat a zpracování zprávy mezi odesílatelem, mezičlánky a konečným příjemcem. Informace v hlavičce mohou být libovolné, ale takové, aby byly pochopeny, pokud je to specifikováno atributem *mustUnderstand="true"*, všemi mezičlánky po cestě. Tělo zprávy nese přenášené informace.

Ukázka SOAP zprávy [19]:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
</env:Header>
<env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
    </p:departure>
    <p:return>
      <p:departing>Los Angeles</p:departing>
      <p:arriving>New York</p:arriving>
    </p:return>
  </p:itinerary>
</env:Body>
</env:Envelope>
```

V hlavičce zprávy je uveden element *reservation*, atribut *role* v tomto případě říká, že se zprávou má zabývat všechny články přenosu, atribut *mustUnderstand* říká, že pokud článek, který se zprávou přijde do styku, nechápe element, musí ukončit zpracovávání zaslat nazpátek Fault (selhání). SOAP definuje způsob pro zaslání Faultu. Hlavička dále obsahuje elementy *reference* a *dateAndTime*, které obsahují dodatečné informace.

Tělo zprávy obsahuje dva objekty – *departure* a *return* s atributy *departing* a *arriving*.

5.3.1 Přenos binárních dat

V případě, že je potřeba ve zprávě přenést binární data, mohou být tato data uložena dvěma způsoby – přímo v těle kódovaná jako Base64 nebo jako binární data mimo tělo zprávy (MTOM).

Při kódování Base64 jsou binární data převedena na tisknutelné textové znaky tak, že se z bitové reprezentace vždy odečte 6 bitů a hodnota těchto bitů odkazuje na určitý symbol v tabulce znaků. Těchto 6 bitů se následně nahradí tímto symbolem. Výsledná zpráva je následně zarovnána na násobek 3B a podle potřeby doplněna 0-2 znaky „=“. Každých 6 bitů se při kódování nahrazuje 8 bity, výsledná zakódovaná data jsou tedy přibližně o 33% větší než původní.

Pokud je využit způsob MTOM (Message Transmission Optimization Mechanism, mechanismus optimalizace přenosu zprávy), jsou data ve zprávě uložena přímo binárně, ale obsahují dodatečnou hlavičku informující o MIME⁴ typu dat, tyto dodatečné informace mohou mít v závislosti na obsahu velikost okolo 400 B – 600 B [20].

Příklad zprávy s MTOM [21] je uveden v příloze P I na konci práce.

Výhodnost jednotlivých řešení vyplývá z množství přenášených binárních dat, pokud je délka binárních dat zanedbatelná (přibližně do 1500 B), vyplatí se data ukládat v kódování Base64, pokud je délka větší, vyplatí se data ukládat ve formátu MTOM. Hodnoty zde jsou pouze orientační.

5.3.2 Vzdálené volání procedur

Nejčastější použití protokolu SOAP je vzdálené volání procedur (RPC, Remote Procedure Call). Při vzdáleném volání procedur je na vzdáleném zařízení zavolána metoda a jsou ji případně předány příslušné parametry. Vzdálené zařízení metodu provede a vrátí její případný výsledek.

⁴ MIME (Multipurpose Internet Mail Extensions) jednoznačně identifikuje formát dat.

Formát zprávy pro vzdálené volání procedury je následující [19]:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Header>
    <t:transaction
      xmlns:t="http://thirdparty.example.org/transaction"
      env:encodingStyle="http://example.com/encoding"
      env:mustUnderstand="true" >5</t:transaction>
  </env:Header>
  <env:Body>
    <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
      xmlns:m="http://travelcompany.example.org/">
      <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
        <m:code>FT35ZBQ</m:code>
      </m:reservation>
      <o:creditCard xmlns:o="http://mycompany.example.com/financial">
        <n:name xmlns:n="http://mycompany.example.com/employees">
          Jan Novák
        </n:name>
        <o:number>123456789099999</o:number>
        <o:expiration>2005-02</o:expiration>
      </o:creditCard>
    </m:chargeReservation>
  </env:Body>
</env:Envelope>
```

V kódu výše se na vzdáleném zařízení volá metoda *changeReservation* a předávají se jí parametry *reservation->code=FT35ZBQ*, *creditCard->name=Jan Novák*, *creditCard->number=123456789099999*, *creditCard->expiration=2005-02*.

Vzdálené zařízení metodu provede a vrátí [19]:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Header>
    <t:transaction
      xmlns:t="http://thirdparty.example.org/transaction"
      env:encodingStyle="http://example.com/encoding"
      env:mustUnderstand="true">5</t:transaction>
  </env:Header>
  <env:Body>
    <m:chargeReservationResponse
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
      xmlns:m="http://travelcompany.example.org/">
      <m:code>FT35ZBQ</m:code>
      <m:viewAt>
        http://travelcompany.example.org/reservations?code=FT35ZBQ
      </m:viewAt>
    </m:chargeReservationResponse>
  </env:Body>
</env:Envelope>
```

5.3.3 Rozšíření WS-*

Protokol SOAP lze rozšířit pomocí specifikací WS (Web Services, webové služby), které za pomoci přidání dalších položek do hlavičky zprávy a / nebo přidání režijních zpráv umožňují komunikaci zabezpečit z hlediska spolehlivosti (potvrzování doručené zprávy, doručení zpráv ve správném pořadí) a bezpečnosti (šifrování přenášených dat, autentizace uživatele). Nutné je, aby byla daná specifikace podporována oběma stranami. Použití WS specifikací může až několikanásobně zvyšovat režii při přenosu.

Specifikace	Použití	Aplikována
WS-Addressing	Umožňuje vkládat směrovací data do SOAP hlavičky.	Položka v hlavičce.
WS-AtomicTransaction	Pokrytuje prostředky pro provedení atomické transakce.	Položka v hlavičce.
WS-Security	Šifrování, podepisování zpráv, ověření identity.	Položka v hlavičce.
WS-MetadataExchange	Umožňuje načíst metadata komunikačního bodu.	Zprávy mimo komunikaci.
WS-CoordinationContext	Koordinace akcí distribuované aplikace.	Zprávy mimo komunikaci.
WS-ReliableMessaging	Umožňuje spolehlivě doručit zprávu, chrání před ztrátou zprávy při chybě sítě.	Předpis pro výměnu zpráv.
WS-Discovery	Definuje multicastový protokol pro nalezení služeb v lokální síti.	Předpis pro výměnu zpráv.
WS-Trust	Vydávání, obnovování a ověřování bezpečnostních tokenů.	Předpis pro výměnu zpráv.
WS-SecureConversation	Vytvoření bezpečnostního kontextu pro více zpráv současně.	Předpis pro výměnu zpráv.
WS-Federation	Umožňuje, aby si různé systémy vzájemně vyměnily informace o identitě uživatele.	Předpis pro výměnu zpráv.

Tab. 1: Přehled vybraných WS specifikací [20]

5.3.4 WSDL

Aby mohl klient zjistit, jaké možnosti (metody, datové objekty) server v rámci služby poskytuje, byl vyvinut jazyk WSDL (Web Services Description Language, jazyk pro popis webových služeb). Použití probíhá tak, že klient udělá dotaz na speciální adresu, ze které si stáhne XML soubor popisující metody, které je možné na službě volat, datové struktury (XML schéma), které se při komunikaci se službou používají a další vlastnosti služby týkající se rozšiřujících WS specifikací a přenosových protokolů (bindings). Tyto informace následně klient použije pro implementaci volání služeb na své straně.

WSDL dokument definuje služby jako kolekce síťových koncových bodů, nebo portů. Ve WSDL, je abstraktní definice koncových bodů a zpráv oddělena od jejich konkrétního síťového nasazení nebo mapování datových formátů. Toto umožňuje znovu využít abstraktní definice: zprávy (messages), které jsou abstraktním popisem vyměňovaných dat a typů portů, jež jsou abstraktními kolekcemi operací. Specifikace konkrétního protokolu a datového formátu pro konkrétní typ portu představuje vícekrát použitelnou vazbu (binding). Port je definován přiřazenou síťovou adresou s vícekrát použitelnou vazbou (binding), kolekce portů definuje službu. [22]

WSDL element používá tyto elementy [22]:

- Types – kontejner pro datové typy.
- Message – abstraktní typová definice pro komunikační data.
- Operation – abstraktní popis akce podporovaný službou.
- PortType – abstraktní sada operací podporovaných jedním nebo více koncovými body.
- Binding – specifikace konkrétního protokolu a datového formátu pro vybraný typ portu.
- Port – jeden koncový bod definovaný jako kombinace bindingu a síťové adresy.
- Service – kolekce příslušných endpointů.

Ukázkový popis služby (WSDL 1.1) [22] je uveden v příloze P II na konci práce.

6 DISTRIBUOVANÉ APLIKACE V .NET

Pro tvorbu distribuovaných aplikací na platformě .NET se aktuálně nejčastěji používají dva frameworky – WebAPI a WCF.

6.1 WebAPI

Technologie WebAPI byla uvedena firmou Microsoft jako součást ASP.NET MVC frameworku v roce 2012 (MVC verze 4) jako jedna ze šablon této technologie. Cílem WebAPI je poskytnout jednoduchý framework pro webové služby postavené na principech REST (viz. kap. 5.1), často je ovšem používána pro vzdálené volání procedur přes protokol HTTP (viz. kap. 5.2).

WebAPI se vyznačuje jednoduchostí implementace. Celou podstatou technologie je na mapovat části URL adresy v souvislosti s metodou (GET, POST, PUT, DELETE) HTTP požadavku na určitou funkci, převzít z URL nebo těla požadavku případné parametry pro tuto funkci, provést funkci a vrátit případné výsledky v požadované reprezentaci specifikované v *Accept* hlavičce HTTP dotazu. Technologie ve výchozím nastavení podporuje vracení výsledků ve formátech XML (viz. kap. 3.1) a JSON (viz. kap. 3.2).

6.1.1 Mapování URL adresy na metodu WebAPI

Pro každý zdroj je vytvořena třída. Sada metod (Vytvořit, Upravit, Získat, Smazat) týkající se daného zdroje se nachází ve třídě, která dědí z *ApiController*. Jméno třídy musí být doplněno řetězcem *Controller*. Ve výchozím nastavení je URL na metody mapována (routování požadavků) tímto způsobem:

```
routeTemplate: "api/{controller}/{id}"
```

kde *{controller}* je jméno třídy, pokud bude na pozici *{id}* v URL hodnota, bude interpretována stejně jako *?id=* a předána do funkce, která očekává parametr *id*.

Při mapování tohoto typu se předpokládá, že metoda ošetřující získání zdroje bude začínat řetězcem *Get*, metoda týkající se vytvoření zdroje bude začínat řetězcem *Post*, metoda týkající se úpravy zdroje bude začínat řetězcem *Put* a metoda smazání zdroje bude začínat řetězcem *Delete*. Metody začínající stejným klíčovým slovem musí být *Public*. Ukázková třída reprezentující zdroj a tento způsob mapování URL je zobrazena níže [23, s. 282]:

```
namespace WebApiSample.Controllers
{
    public class ValuesController : ApiController {
        // GET api/values
        public IEnumerable<string> Get() {
            return new string[] { "value1", "value2" };
        }
        // GET api/values/5
        public string Get(int id) {
            return "value";
        }
        // POST api/values
        public void Post([FromBody] string value) {
        }
        // PUT api/values/5
        public void Put(int id, [FromBody] string value) {
        }
        // DELETE api/values/5
        public void Delete(int id) {
        }
    }
}
```

Na příkladu si lze povšimnout anotace *[FromBody]*, toto frameworku říká, že má danou hodnotu načíst z těla dotazu a ne z URL. Pokud tato anotace není uvedena, jsou hodnoty primitivních datových typů automaticky přebírány z URL a komplexních datových typů (třídy) přebírány z těla požadavku.

Routování HTTP požadavků lze upravit na tvar [24]:

```
routeTemplate: "api/{controller}/{action}/{id}"
```

Za parametr *{action}* lze přímo dosadit název metody. Třidu lze dále upravit tak, aby se metody mohly jmenovat libovolně. Přiřazení HTTP požadavku k metodě potom probíhá pomocí anotací v *[]* a dodaných parametrů metody. Ukázka řešení je uvedena níže:

```
namespace WebApiSample.Controllers
{
    public class ValuesController : ApiController {
        // GET api/values/Data1/5
        [HttpGet]
        [ActionName("Data1")]
        public string LoadData1(int id) {
```

```
        return "value";
    }

    // POST api/values/Data1
    [HttpPost]
    [ActionName("Data1")]
    public void StoreData1([FromBody] string value) {
    }

    // GET api/values/Data2/5
    [HttpGet]
    [ActionName("Data2")]
    public string LoadData2(int id) {
        return "value";
    }
    ...
}
}
```

Z výše uvedeného kódu je patrné, že lze libovolnou Public metodu poskytovat pod jakýmkoliv jménem a jakoukoliv HTTP metodou (z funkčního hlediska to ale musí mít smysl).

6.1.2 Rozšiřitelnost

WebAPI lze rozšířit o vlastní Message Handlers, MediaTypeFormatters a Atributy.

Message Handler je třída, která přijímá HTTP dotaz a vrací HTTP odpověď. Message Handlers dědí z abstraktní třídy *HttpMessageHandler*. Message Handlers dokáží určitým způsobem modifikovat požadavek ještě před tím, než dorazí na Controller, případně odpověď potom, co opustí Controller. Typické případy použití jsou [25]:

- Úprava hlavičky požadavků.
- Přidání hlavičky do odpovědi.
- Ověření (povolení/zakázání) požadavku dříve než dosáhne Controller.

MediaTypeFormatter je třída starající se o serializaci a deserializaci třídy do / z přenášené zprávy. Formát je identifikován MIME typem. MessageFormatters dědí ze třídy *BufferedMediaTypeFormatter*. [26]

Atributy informují framework o tom, jak má s daným prvkem (třídou nebo metodou) pracovat (umísťují se do [] nad prvek, kterého se týkají), mohou být různého typu, například *AuthorizeAttribute* umožňuje implementovat vlastní autentizaci uživatelů – k prvkům označeným tímto atributem mohou přistupovat jen autentizovaní uživatelé.

6.2 WCF

WCF (Windows Communication Foundation, základ komunikace Windows) je unifikovaný framework pro operační systém Microsoft Windows, který slouží ke vzdálenému volání procedur na jiném zařízení. Procedury jsou přístupné skrze tzv. služby. Předností frameworku je kombinace jedné implementace (jeden kód) služby a mnoha způsobů, jak je službě přistupovat.

Pro popis zde bude WCF rozděleno na tři části: interface služby, implementaci služby a binding.

6.2.1 Interface

Interface (rozhraní) popisuje službu z hlediska práce s ní. Specifikuje názvy metod, parametry metod a návratové typy. Dále je zde možno pomocí datových anotací specifikovat přístup ke službě a metadata. Interface je využit při tvorbě WSDL (viz. kap. 5.3.4) popisu služby.

Ukázka interface je níže [3], zde je nutno si povšimnout anotace *[ServiceContract]*, která říká, že je interface zpřístupněn jako služba a *[OperationContract]*, která říká, že se daná metoda účastní služby. V anotacích lze uvést další parametry, například *[ServiceContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]* vynucuje, aby byla služba poskytována pouze přes protokol, který šifruje a podepisuje zprávy, *[OperationContract(IsOneWay=true)]* říká, že metoda nevrací žádnou odpověď (ani výjimku, která nastala při zpracování metody na serveru).

```
[ServiceContract]
public interface IHelloWorld
{
    [OperationContract]
    string SayHello(Name person);
}
```

6.2.2 Implementace služby

Služba je implementována ve třídě, která implementuje interface služby. Třída nemusí být označena žádnou anotací, anotace zde jsou volitelné a slouží pro běhové prostředí. Například `[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]` nad třídou říká, že má na serveru existovat pouze jedna společná instance služby pro všechny klienty. `[OperationBehavior(Impersonation=ImpersonationOption.Required)]` nad metodou říká, že má být metoda provedena s právy uživatele, který ji vzdáleně volá (uživatel musí být přihlášen). Ukázka implementace služby je níže [3]:

```
public class HelloWorldService : IHelloWorld
{
    public string SayHello(Name person)
    {
        return string.Format("Hello {0} {1}", person.First, person.Last);
    }
}
```

Metoda přijímá datový objekt *Name* a vrací textový řetězec. Objekty, které se účastní komunikace, musí být označeny anotací `[DataContract]` a pole v objektech, které se přenášejí anotací `[DataMember]`. Tyto anotace používá *DataContractSerializer* (část .NET) při serializování a deserializování objektu do / ze zprávy. Serializace a deserializace objektu do / ze zprávy lze ovlivnit parametry anotací. Například `[DataContract(Name="NameAndSurname")]` říká, že objekt třídy *Name* bude ve zprávě pojmenován jako *NameAndSurname* a `[DataMember(IsRequired=true)]` vyžaduje, aby byla hodnota vždy vyplněna (toto souvisí se schématem XML dokumentu – viz. kap. 3.1.4). Ukázka anotací třídy je uvedena níže [3]:

```
[DataContract]
public class Name
{
    [DataMember]
    public string First;
    [DataMember]
    public string Last;
}
```

6.2.3 Komunikační bod

Služba je navenek nabízena přes komunikační bod (service endpoint). K tomuto bodu se připojují klienti využívající službu.

Komunikační bod se skládá ze tří částí [27 s. 46]:

- Adresy, na které je služba dostupná.
- Bindingu, který říká, přes který protokol je služba dostupná.
- Kontraktu, který definuje interface služby (viz. kap. 6.2.1).

Klient si pomocí staženého WSDL dokumentu vytvoří potřebnou strukturu pro komunikaci se službou, vybere si koncový bod, se kterým chce komunikovat a nastaví parametry komunikace pomocí parametrů (Adresa, Binding, Kontrakt) tohoto bodu. Klient vytvoří tzv. Channel (kanál) k tomuto bodu.

6.2.4 Binding

Binding (vazba) udává, přes jaký protokol bude služba zpřístupněna. WCF podporuje několik základních bindingů. Nejdůležitější jsou uvedeny v tabulce níže:

Jméno Bindingu	Způsob použití Bindingu
WebHttpBinding	Kompatibilní komunikace založená na principech REST přes protokol HTTP.
BasicHttpBinding	Kompatibilní komunikace založená na principech SOAP přes protokol HTTP, ve výchozím nastavení neposkytuje ani šifrování, ani spolehlivé doručování zpráv.
WSHttpBinding	Kompatibilní komunikace založená na principech SOAP přes protokol HTTP umožňuje používat řadu WS-* (viz. kap. 5.3.3) rozšíření (šifrování zpráv, spolehlivé doručování apod.).
NetTcpBinding	Komunikace mezi zařízeními používajícími .NET, zprávy jsou serializovány binárně a přenášeny přes protokol TCP.
NetPeerTcpBinding	Komunikace mezi zařízeními používajícími .NET, po vzájemném vyhledání zařízení přes spojovací službu je pro přenos zpráv využit princip P2P.
NetNamedPipesBinding	Mezi-procesová komunikace v rámci jednoho zařízení pomocí jmenových rour.
NetMsmqBinding	Synchronní spolehlivá komunikace pomocí fronty zpráv. Pro komunikaci je využit protokol MSMQ ⁵ .

Tab. 2: Přehled nejdůležitějších bindingů WCF

⁵MSMQ (Microsoft Message Queuing) je implementace principu fronty zpráv firmou Microsoft. Komunikace probíhá asynchronně, technologie umožňuje komunikaci mezi aplikacemi běžícími v rozdílných časech. Úprava komunikace spočívá v zavedení mezi-prvku, který běží neustále. Odesílatel pošle zprávu mezi-prvku, který ji zařadí do fronty zpráv pro daného příjemce. Zpráva je příjemci doručena buďto okamžitě, nebo po tom, kdy je schopen zprávu přijmout. Přijetí zprávy je možno potvrzovat a zabezpečit tak spolehlivou komunikaci [28].

Jednotlivé bindingy mají další možnosti konfigurace, které jsou po ně specifické. Například `BasicHttpBinding` a `WsHttpBinding` umožňují mimo jiné nastavit kódování zpráv na `Text/MTOM` (viz. kap. 5.3.1). `NetTcpBinding` umožňuje nastavit maximum TCP spojení, které služba vytváří a u `WebHttpBindingu` lze nastavit kódování textu, kterým jsou zprávy zapisovány.

Konfigurace koncového bodu může probíhat buď v kódu nebo v XML souboru s nastavením služby [27, s. 81]. Ukázka konfigurace v XML souboru s nastavením služby je uvedena níže:

```
<system.serviceModel>
  <bindings>
    <basicHttpBinding>
      <binding name="NewBinding0" closeTimeout="00:02:00" />
    </basicHttpBinding>
  </bindings>
  <services>
    <service name="WCFSimple.HelloWorldService">
      <endpoint address="http://localhost:8080/" binding="basicHttpBinding"
        bindingConfiguration="" contract="WCFSimple.IHelloWorld" />
    </service>
  </services>
</system.serviceModel>
```

V kódu výše je nakonfigurována služba `WCFSimple.HelloWorldService`, tak, že je zpřístupněna přes `BasicHttpBinding` na adrese `http://localhost:8080/`, služba je popsána interfacem `WCFSimple.IHelloWorld`. `BasicHttpBinding` má upraven parametr (změna oproti výchozím hodnotám) `closeTimeout` na hodnotu `00:02:00`.

6.3 WebSocket

Protokol `WebSocket` byl schválen IETF (Internet Engineering Task Force) teprve v prosinci 2011. Nejedná se o klasický protokol pro distribuované aplikace, ale díky jeho budoucímu potenciálu se jej sluší zde krátce zmínit. Podstatou `WebSocket` protokolu je umožnit webovému prohlížeči provést dotaz na server přes socketové propojení podobné `TCP`, bez použití protokolu `HTTP`.

V dnešní době je dynamická změna obsahu na webové stránce často řešena pomocí asynchronních dotazů (`AJAX`), kdy se webový prohlížeč periodicky v daných intervalech

dotazuje serveru na aktuální hodnotu dat. Toto s sebou přináší nevýhodu zbytečné reže, protože se klient dotazuje i tehdy, kdy ke změně dat nedošlo, k dotazům se používá protokol HTTP, který při každém dotazu a odpovědi posílá relativně objemnou hlavičku.

WebSocket je lehká nadstavba přímo nad protokolem TCP (viz. kap. 4.1), která přes TCP umožňuje plně duplexně posílat zprávy libovolné délky (zprávy jsou složeny z rámců) přímo z webového prohlížeče. Použití protokolu WebSocket je následující. Webová stránka řekne pomocí JavaScriptu webovému prohlížeči, že má navázat WebSocket spojení s daným serverem. Potom, co je spojení navázáno, může se stránka přes spojení dotazovat serveru na změny dat bez reže protokolu HTTP. Celý koncept může jít ještě dále – stránka může pouze pasivně (s otevřeným spojením) čekat, než ji server sám pošle informace a tyto po přijetí pouze zobrazí (zavolá se event. handler v JavaScriptu, rozparsuje přijatá data a zobrazí je do příslušných částí webové stránky). Adresa protokolu začíná prefixem `ws://`.

Protokol zavádí pouze minimální režii: každý rámeček obsahuje informaci, zda přenáší binární data nebo textová data kódovaná v UTF-8 a o své velikosti. Protokol může rovněž zasílat režijní informace nutné k udržení spojení, velikost těchto informací je ovšem vzhledem k hlavičce protokolu HTTP zanedbatelná.

Pro navázání WebSocket spojení se používá standardní http hlavička, ve které se specifikuje vůle přejít na WebSocket spojení, druhá strana přechod na WebSocket buď potvrdí pomocí HTTP hlavičky s odpovědí, nebo zamítne. HTTP hlavičky jsou tu voleny z důvodu zmatení zařízení po cestě, aby si myslely, že se přes TCP navazuje standardní HTTP spojení.

Handshake je následující:

Klient otevře TCP spojení a pošle serveru [29]:

```
GET /resource HTTP/1.1\r\n
host: example.com\r\n
upgrade: websocket\r\n
connection: upgrade\r\n
sec-websocket-version: 13\r\n
sec-websocket-key: E4WSEcseoWr4csPLS2QJHA==\r\n
\r\n
```

Hlavička *sec-websocket-key* je náhodné 16 bytové číslo zakódované do Base64.

Server odpoví [29]:

```
HTTP/1.1 101 OK
upgrade: websocket\r\n
connection: upgrade\r\n
sec-websocket-accept: 7eQChgCtQMnVILefJA06dK5JwPc=\r\n
\r\n
```

Hlavička *sec-websocket-accept* je vytvořena tak, že je k řetězci z hlavičky *sec-websocket-key* přidán řetězec *258EAF45-E914-47DA-95CA-C5AB0DC85B11*, výsledný řetězec je hashován funkcí SHA1 a převeden do Base64 reprezentace. [30]

Stejně jako HTTPS, lze protokol provozovat přes TLS spojení, zde adresa začíná *wss://*. Protokol je momentálně dostupný pouze na Windows 8 a Windows Server 2012.

7 PROSTŘEDKY PRO ANALÝZU A ÚPRAVU SÍŤOVÉ KOMUNIKACE

V této kapitole je krátký popis prostředků použitých pro analýzu síťové komunikace, které byly aktivně využity při zpracování diplomové práce.

7.1 Wireshark

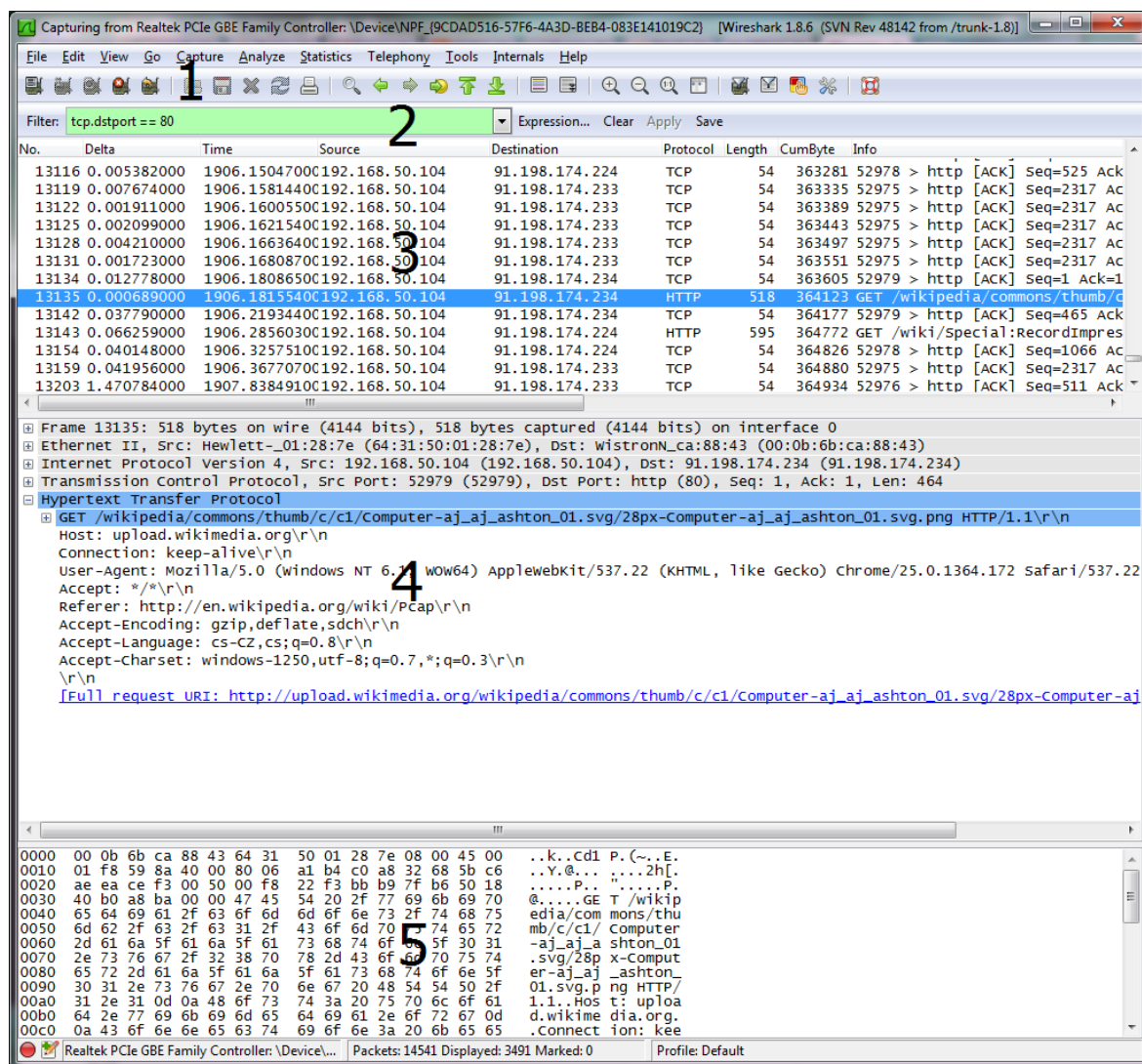
Wireshark je paketový sniffer s funkcionalitou analýzy síťových protokolů. Je šířen pod GNU GPL⁶ licenci, nejčastěji se využíván pro řešení síťových problémů.

Jedná se o okenní aplikaci, která umožňuje vybrat síťový adaptér, spustit odposlouchávání komunikace na tomto adaptéru pomocí rozhraní *Pcap* a dále zachycenou komunikaci zobrazit. Zachycená komunikace lze filtrovat dle určitých hledisek (protokol na vyšší vrstvě, zdrojová IP adresa, cílová IP adresa, port apod.), po výběru určité části komunikace ze střední části okna se ve spodní části okna zobrazí podrobnosti této části komunikace pro jednotlivé protokoly a čistá binární data v hexadecimální nebo binární reprezentaci. Průběh vybrané TCP komunikace lze sledovat pomocí volby „Follow TCP Stream“ (následovat TCP proud). Komunikaci lze uložit do souboru a následně později načíst a pracovat s ní stejně, jako by byla aktuálně zachycená. [31]

Na obrázku níže je popsáno okno programu Wireshark:

1. Ovládací panel.
2. Aplikace filtrů.
3. Zachycená komunikace (filtrovaná dle filtru v bodě 2).
4. Podrobnosti o daném fragmentu komunikace.
5. Surová data v hexadecimální reprezentaci.

⁶ GNU GPL (General Public Licence, všeobecná veřejná licence) dává uživatelům softwaru právo jej svobodně užívat a podle potřeby dále upravovat. Pokud chtějí software dále distribuovat, mohou si vyžádat poplatek, ale musí jej distribuovat se všemi zdrojovými kódy zase pod licenci GNU.



Obr. 6: Okno programu Wireshark

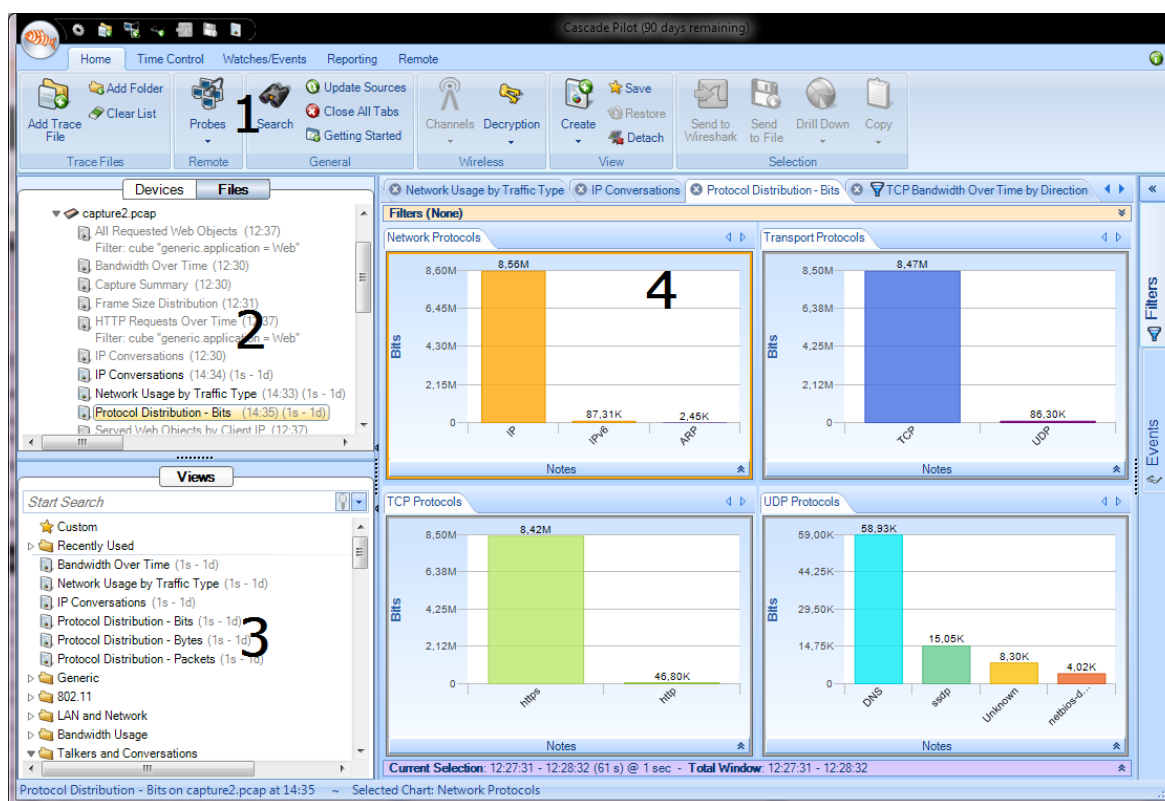
7.2 Cascade Pilot

Cascade Pilot je komerční aplikace firmy Riverbed. Pracuje se stejným formátem souborů pro zachycená data jako program Wireshark a umožňuje s ním spolupracovat. Na rozdíl od Wiresharku ale neumožňuje s daty pracovat přímo tak, jak jsou. Místo toho nabízí několik pohledů, kterými se dá na zachycenou komunikaci jako celek (nebo dílče) dívat. Komunikace tak lze zanalyzovat a následně graficky vizualizovat z hlediska použitých protokolů a parametrů těchto protokolů. Například lze sledovat rychlost komunikace v průběhu času, využití přenosového pásma v závislosti na jednotlivých protokolech, zobrazení všech probíhajících komunikací v daném časovém rámci, sledování změn

velikosti TCP okna v průběhu času apod. Grafické průběhy lze získat jednoduše přetažením vybraného *View* na daný načtený soubor (zachycenou komunikaci). Z grafických průběhů lze následně vygenerovat report ve formátu PDF.

Popis okna programu Cascade Pilot:

1. Ovládací panel.
2. Výběr zachycené komunikace.
3. Views, pomocí kterých se dá komunikace vizualizovat.
4. Vizualizovaná komunikace.



Obr. 7: Okno programu Cascade Pilot

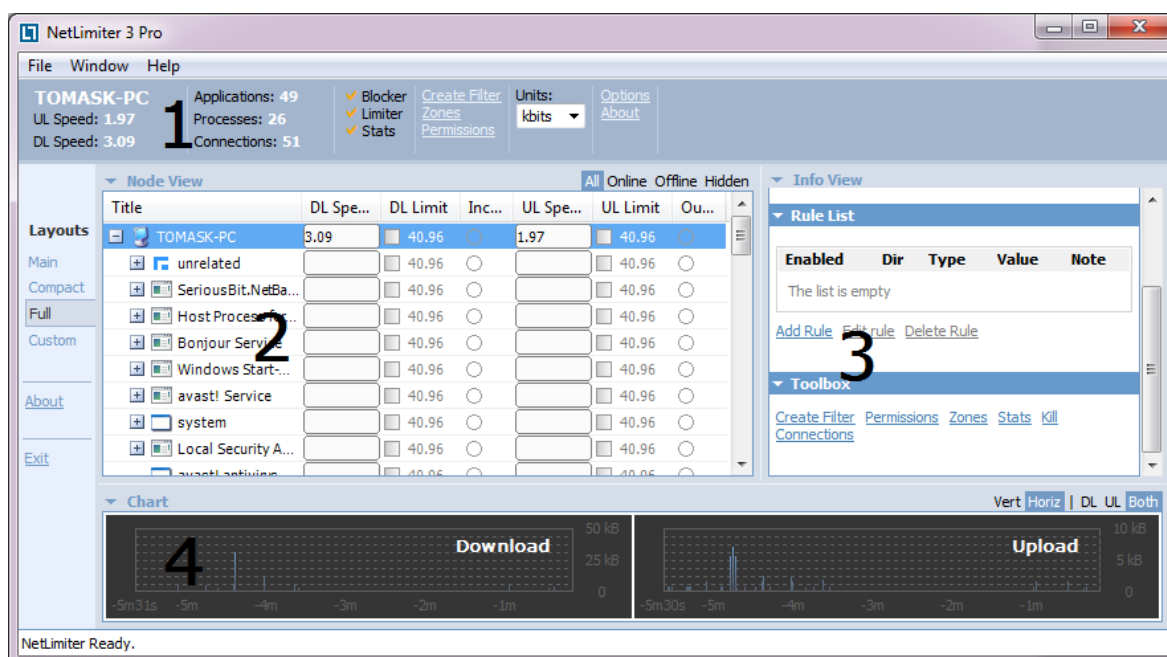
7.3 NetLimiter

Program NetLimiter je komerční software vyvinutý společností Locktime Software určený k omezení rychlosti síťové komunikace jednotlivých procesů daného počítače. Pro každý proces lze nastavit maximální rychlosti, jakými může data přijímat a odesílat, komunikace procesu lze úplně zablokovat, program rovněž umožňuje vytvářet časová pravidla

a limitovat komunikaci v závislosti na dnu a čase. Program rovněž umožňuje sledovat rychlost síťové komunikace v reálném čase a to jak pro celý počítač, tak pro jednotlivé komunikující procesy.

Popis okna programu Netlimiter 3.0 Pro:

1. Souhrnné aktuální informace.
2. Přehled procesů, aktuální komunikační rychlosti daného procesu, možnost omezit maximální rychlost stahování / odesílání dat (nezávisle na sobě) pro daný proces.
3. Přehled aktuálně používaných pravidel pro daný proces, možnost zobrazení statistik.
4. Grafické znázornění síťové komunikace daného procesu / celého počítače za posledních 5 - 15 minut (dle roztažení okna).



Obr. 8: Okno programu NetLimiter

II. PRAKTICKÁ ČÁST

8 ZÁMĚRY TESTU

Záměrem testů je ověřit vliv různých způsobů komunikace a datových formátů na dobu přenosu dat při konstantní propustnosti sítě. Přenášejí se vždy stejná data, ale pokaždé jiným způsobem. Cílem je sestavit přehledné srovnání jednotlivých způsobů komunikace, z hlediska času, který uplyne od okamžiku klientova odeslání požadavku na službu do okamžiku příchodu odpovědi od služby a na základě výsledků měření doporučit nejvhodnější způsoby přenosu pro splnění požadované funkcionality síťové komunikace.

Čas je měřen z hlediska klienta a zahrnuje:

- Čas potřebný na sestavení požadavku na klientovi.
- Čas potřebný na přenos požadavku klienta na server.
- Čas potřebný na zpracování požadavku a vytvoření odpovědi na straně serveru.
- Čas potřebný na doručení odpovědi serveru klientovi.
- Čas potřebný na zpracování odpovědi serveru na klientovi.

Důležitý je zde právě onen fakt, že se měří celkový čas od požadavku klienta na službu do doručení odpovědi od služby, je tak možno vidět rychlost komunikace jednotlivých způsobů / protokolů jako celek a brát v potaz nejen rychlost přenosu dat po síti, ale rovněž i čas potřebný na serializaci/deserializaci přenášených dat. Toto značně zvyšuje relevantnost testů oproti testům, kde se měří pouze serializace / počet volání dané funkce za konstantní čas přes různé protokoly, protože tyto testy, leč velmi oblíbené, nemají téměř žádnou vypovídající hodnotu z hlediska praktického používání služeb, protože netestují různé množství dat serializované v různých formátech přes různé protokoly při omezené propustnosti sítě.

Aby byly naměřené hodnoty názornější a rozestup mezi jednotlivými způsoby přenosu viditelnější, jsou přenosy dat měřeny na linkách se symetrickými rychlostmi 16 kbit/s, tj. 2 kB/s, 128 kbit/s, tj. 16 kB/s a 10 Mbit/s, tj. 1250 kB/s. Výsledky přenosu menšího množství dat jsou následně vyhodnocovány z nižší přenosové rychlosti, výsledky přenosu větších objemů dat jsou vyhodnocovány z vyšší přenosové rychlosti.

Testy jsou provedeny opakovaně a výsledky statisticky zpracovány.

8.1 Základní rozdělení testů

Testy jsou rozděleny na tři základní části:

- Test vlivu serializace objektu do formátů JSON, XML, SOAP.
- Test vlivu rozdílných Bindingů / rozdílného nastavení parametrů bindingu v technologii WCF.
- Test vlivu různých možností při uploadu souboru na server.

8.1.1 Test vlivu serializace objektu do formátů JSON, XML, SOAP

Princip testu spočívá v tom, že klient stahuje ze serveru objekt s měnícím se počtem vnitřních proměnných, objekt je postupně serializován do formátů JSON, XML a SOAP. Využita je technologie WebAPI a rozdílné Bindingy a rozdílné parametry daných Bindingů technologie WCF. Objekt obsahuje tento počet proměnných: 1, 5, 10, 20, 40. Cílem je pozorovat vliv složitosti objektu a způsobu serializace na čas odezvy.

8.1.2 Test vlivu použité technologie a formátu na dobu stahování binárních dat

Princip testu spočívá v tom, že klient ze serveru stahuje určité předem dané množství binárních dat - 1B, 5B, 10B, 50B, 100B, 500B, 1kB, 5kB, 10kB, 50kB, 100kB, 500kB, 1MB, 5MB, 10 MB a to jak v objektu serializovaném do JSON/XML, tak i pomocí různých Bindingů / parametrů daného Bindingu technologie WCF. Měří se doba přenosu těchto dat.

8.1.3 Test vlivu různých možností při uploadu souboru na server

Princip testu spočívá v uploadu binárních dat o velikostech 500kB, 1MB, 5MB, 10MB na sever, data jsou přenášena na WebAPI server jako parametr metody (kódovaná base64), jako MIME binární data, dále pak pomocí technologie WCF, kde je kladen důraz na zjištění, jak se časově projeví přenos pomocí SOAP zpráv, které jsou zpracovány na principu Streamingu a Bufferingu za použití protokolů HTTP a TCP.

9 ZPŮSOB PROVEDENÍ TESTŮ

Testy byly provedeny v laboratorním i reálném prostředí (na internetu). Další popis se týká testů v laboratorním prostředí. Specifika testování v prostředí internetu budou probrány zvlášť na konci kapitoly.

Test probíhal na síti LAN s propustností 1 Gbit/s. Skutečná rychlost přenosu dat byla upravena pomocí programu NetLimiter (viz. kap. 7.3). Rychlost přenosu dat byla volena na základě objemu přenášených dat a byla vždy shodná pro všechny způsoby přenosu daných dat. Rychlost byla měněna skokově 16 kbit/s, 128 kbit/s, 10 Mbit/s. Data byla dále zachytávána pomocí programu Wireshark (viz. kap. 7.1).

Poznámka k úpravě rychlosti:

Z hlediska měření času potřebného na přenos dat nemá smysl přenášet 100 B dat rychlostí 10 Mbit/s, protože (teoretická) doba trvání přenosu (přibližně 85 μ S) je pod měřitelnou úrovní, stejně tak nemá smysl přenášet 10 MB dat rychlostí 16 kbit/s, protože při využití 13 různých způsobů by jeden test trval 130 hodin, nehledě na to, že je nutno test mnohokrát opakovat pro ověření správnosti výsledku. Pro dané množství dat byla tedy zvolena smysluplná rychlost tak, aby se výsledky naměřeného času na jeden přenos pohybovaly v rozmezí 100 ms až 10 s.

Poznámka k množství přenášených dat:

Po snadnější přepočty byl zvolen dekadický násobitel, tj. přípona k (kilo) znamená násobení hodnotou 1 000 a přípona M (mega) násobení hodnotou 1 000 000. 10 MB dat tedy znamená 10 000 000 B, tento fakt ulehčuje časové přepočty mezi jednotlivými řády, protože 100 B je 50x méně než 5 kB apod. Rychlost je rovněž limitována v násobcích 1000, tj. 10 Mbit/s znamená 10 000 000 bitů/s, tj. 1 250 000 B/s, zapsáno jako 1,25 MB/s. Pokud není uvedeno jinak, jsou všechny testované velikosti přenášených dat uvedeny na aplikační vrstvě.

9.1 Testy stahování dat

Pro testy týkající se serializace objektu (viz. 8.1.1) a stahování binárních dat (viz. 8.1.2) byla využita kombinace každý s každým, tj. všechny jednotky dat, které jsou uvedeny níže,

byly přenášeny všemi způsoby, které jsou uvedeny níže s výjimkou bindingu UDP (WCF), kde byla maximální velikost přenášených dat 10kB.

9.1.1 Přehled přenášených jednotek dat

Tabulka níže obsahuje kódové označení (identifikaci) jednotky dat, popis jednotky dat a rychlost, při které byl čas přenosu jednotky měřen.

Identifikace	Popis	Přenášeno na rychlosti
Format1	Objekt s 1 vnitřní proměnnou.	16 kbit/s
Format5	Objekt s 5 vnitřními proměnnými.	16 kbit/s
Format10	Objekt s 10 vnitřními proměnnými.	16 kbit/s
Format20	Objekt s 20 vnitřními proměnnými.	16 kbit/s
Format40	Objekt s 40 vnitřními proměnnými.	16 kbit/s
Binary1	Objekt s 1 polem o velikosti 1 B.	16 kbit/s
Binary5	Objekt s 1 polem o velikosti 5 B.	16 kbit/s
Binary10	Objekt s 1 polem o velikosti 10 B.	16 kbit/s
Binary50	Objekt s 1 polem o velikosti 50 B.	16 kbit/s
Binary100	Objekt s 1 polem o velikosti 100 B.	16 kbit/s
Binary500	Objekt s 1 polem o velikosti 500 B.	16 kbit/s
Binary1k	Objekt s 1 polem o velikosti 1 000 B.	16 kbit/s
Binary5k	Objekt s 1 polem o velikosti 5 000 B.	128 kbit/s
Binary10k	Objekt s 1 polem o velikosti 10 000 B.	128 kbit/s
Binary50k	Objekt s 1 polem o velikosti 50 000 B.	128 kbit/s
Binary100k	Objekt s 1 polem o velikosti 100 000 B.	128 kbit/s
Binary500k	Objekt s 1 polem o velikosti 500 000 B.	10 Mbit/s
Binary1M	Objekt s 1 polem o velikosti 1 000 000 B.	10 Mbit/s
Binary5M	Objekt s 1 polem o velikosti 5 000 000 B.	10 Mbit/s
Binary10M	Objekt s 1 polem o velikosti 10 000 000 B.	10 Mbit/s

Tab. 1: Seznam datových jednotek pro směr ke klientovi – laboratorní test

9.1.2 Přehled způsobů přenosu dat

Tabulka níže obsahuje způsoby, kterými byly datové jednotky z *Tab. 1* přenášeny.

Identifikace	Popis
WebAPI-JSON	Technologie WebAPI, objekt serializován do formátu JSON (HTTP).
WebAPI-XML	Technologie WebAPI, objekt serializován do formátu XML (HTTP).
basicHttpBinding-default	Technologie WCF, basicHttpBinding s výchozím nastavením (HTTP).
basicHttpBinding-transportSecurity	Technologie WCF, basicHttpBinding se zabezpečením na úrovni transportní vrstvy (HTTPS).
basicHttpBinding-mtom	Technologie WCF, basicHttpBinding s přenosem binárních dat ve formátu MTOM (HTTP).
wsHttpBinding-default	Technologie WCF, wsHttpBinding se zabezpečením na úrovni zpráv Message Security (HTTP).
wsHttpBinding-transportSecurity	Technologie WCF, wsHttpBinding se zabezpečením na úrovni transportní vrstvy (HTTPS).
wsHttpBinding-noSecurity	Technologie WCF, wsHttpBinding se Security nastaveno na None (HTTP).
wsHttpBinding-reliableNoSecurity	Technologie WCF, wsHttpBinding se Security nastaveno na None a aktivovaným Reliable Messagingem s garantovaným doručováním zpráv v pořadí odeslání (HTTP).
wsHttpBinding-defaultWithReliable	Technologie WCF, wsHttpBinding se zabezpečením na úrovni zpráv Message Security a aktivovaným Reliable Messagingem s garantovaným doručováním zpráv v pořadí odeslání (HTTP).
netTcpBinding-default	Technologie WCF, netTcpBinding se zabezpečením na úrovni transportní vrstvy (Binární TCP).
netTcpBinding-defaultWithReliable	Technologie WCF, netTcpBinding se zabezpečením na úrovni transportní vrstvy a aktivovaným Reliable Messagingem s garantovaným doručováním zpráv v pořadí odeslání (Binární TCP).
netTcpBinding-default	Technologie WCF, netTcpBinding bez zabezpečení na úrovni transportní vrstvy (Binární TCP).

Tab. 2: Seznam způsobů přenosu dat pro směr ke klientovi – laboratorní test

9.2 Testy odesílání dat

Testy odesílání dat (viz. 8.1.3) se týká specifiky odesílání binárních dat z klienta na server, které nastává v případě uploadu souboru. Tabulky níže obsahují datové jednotky, které byly přenášeny a způsoby, přes které byly přenášeny.

9.2.1 Přehled přenášených jednotek dat

Identifikace	Popis	Přenášeno na rychlosti
Upload500k	Bytové pole o velikosti 500 000 B.	10 Mbit/s.
Upload1M	Bytové pole o velikosti 1 000 000 B.	10 Mbit/s.
Upload5M	Bytové pole o velikosti 5 000 000 B.	10 Mbit/s.
Upload10M	Bytové pole o velikosti 10 000 000 B.	10 Mbit/s.

Tab. 3: Seznam datových jednotek pro směr od klienta – laboratorní test

9.2.2 Přehled způsobů přenosu dat

Tabulka níže obsahuje způsoby, kterými byly datové jednotky z Tab. 3 přenášeny.

Identifikace	Popis
WebAPI-asParameter	Technologie WebAPI, pole předáno jako parametr metody, přenášeno ve formátu Base64 (HTTP).
WebAPI-multipart	Technologie WebAPI, pole přenášeno jako Multipart/form-data. MIME typ binary/octet-stream (HTTP).
basicHttpBinding-streaming	Technologie WCF, basicHttpBinding, TransferMode nastaveno na Streaming (HTTP).
basicHttpBinding-streamingMTOM	Technologie WCF, basicHttpBinding, TransferMode nastaveno na Streaming, pro přenos binárních dat použit MTOM (HTTP).
basicHttpBinding-buffering	Technologie WCF, basicHttpBinding, TransferMode nastaveno na Buffering (HTTP).
netTcpBinding-streaming	Technologie WCF, netTcpBinding, TransferMode nastaveno na Streaming (Binární TCP).
netTcpBinding-buffering	Technologie WCF, netTcpBinding, TransferMode nastaveno na Buffering (Binární TCP).

Tab. 4: Seznam způsobů přenosu dat pro směr od klienta – laboratorní test

9.3 Rozdíl laboratorní a Internetové verze

Při vytváření veřejné Internetové verze testu bylo nutno vzít v potaz fakt, že test je dobrovolný a pro širokou veřejnost, původní 8 hodinový laboratorní test se tedy musel výrazně zkrátit tak, aby jej byli lidé ochotni provést. Dále potom, na rozdíl od kontrolovaného laboratorního prostředí nelze u veřejného testu přes Internet zaručit konstantní komunikační rychlost ani odezvu sítě v průběhu celého testu. Rychlost připojení k Internetu u různých uživatelů může být rovněž rozmanitá. Vypovídací hodnota veřejného testu je tedy mnohem nižší než testu laboratorního.

Po pečlivé úvaze byl test zkrácen na 15 minut a to vypuštěním stahování dat menších než 50 kB a větších než 1 MB (nižší velikosti nemá přes Internet smysl měřit, vyšší by test

znatelně protáhly). Byl rovněž vypuštěn test serializace objektu. Velikost uploadovaného souboru byla volena v rozmezí 50 kB až 1 MB.

Test probíhal všemi způsoby, jako test laboratorní s výjimkou udpBindingu, který byl vynechán.

9.3.1 Stahování dat

Identifikace	Popis	Přenášeno na rychlosti
Binary50k	Objekt s 1 polem o velikosti 50 000 B.	Dle rychlosti uživatele
Binary100k	Objekt s 1 polem o velikosti 100 000 B.	Dle rychlosti uživatele
Binary500k	Objekt s 1 polem o velikosti 500 000 B.	Dle rychlosti uživatele
Binary1M	Objekt s 1 polem o velikosti 1 000 000 B.	Dle rychlosti uživatele

Tab. 5: Seznam datových jednotek pro směr ke klientovi – veřejný test

Identifikace	Popis
WebAPI-JSON	Technologie WebAPI, objekt serializován do formátu JSON (HTTP).
WebAPI-XML	Technologie WebAPI, objekt serializován do formátu XML (HTTP).
basicHttpBinding-default	Technologie WCF, basicHttpBinding s výchozím nastavením (HTTP).
basicHttpBinding-transportSecurity	Technologie WCF, basicHttpBinding se zabezpečením na úrovni transportní vrstvy (HTTPS).
basicHttpBinding-mtom	Technologie WCF, basicHttpBinding s přenosem binárních dat ve formátu MTOM (HTTP).
wsHttpBinding-default	Technologie WCF, wsHttpBinding se zabezpečením na úrovni zpráv Message Security (HTTP).
wsHttpBinding-transportSecurity	Technologie WCF, wsHttpBinding se zabezpečením na úrovni transportní vrstvy (HTTPS).
wsHttpBinding-noSecurity	Technologie WCF, wsHttpBinding se Security nastaveno na None (HTTP).
wsHttpBinding-reliableNoSecurity	Technologie WCF, wsHttpBinding se Security nastaveno na None a aktivovaným Reliable Messagingem s garantovaným doručováním zpráv v pořadí odeslání (HTTP).
wsHttpBinding-defaultWithReliable	Technologie WCF, wsHttpBinding se zabezpečením na úrovni zpráv Message Security a aktivovaným Reliable Messagingem s garantovaným doručováním zpráv v pořadí odeslání (HTTP).
netTcpBinding-default	Technologie WCF, netTcpBinding se zabezpečením na úrovni transportní vrstvy (Binární TCP).
netTcpBinding-defaultWithReliable	Technologie WCF, netTcpBinding se zabezpečením na úrovni transportní vrstvy a aktivovaným Reliable Messagingem s garantovaným doručováním zpráv v pořadí odeslání (Binární TCP).

Tab. 6: Seznam způsobů přenosu dat pro směr ke klientovi – veřejný test

9.3.2 Odesílání dat

Identifikace	Popis	Přenášeno na rychlosti
Upload50k	Bytové pole o velikosti 50 000 B.	Dle rychlosti uživatele
Upload100k	Bytové pole o velikosti 100 000 B.	Dle rychlosti uživatele
Upload500k	Bytové pole o velikosti 500 000 B.	Dle rychlosti uživatele
Upload1M	Bytové pole o velikosti 1 000 000 B.	Dle rychlosti uživatele

Tab. 7: Seznam datových jednotek pro směr od klienta – veřejný test

Identifikace	Popis
WebAPI-AsParameter	Technologie WebAPI, pole předáno jako parametr metody, přenášeno ve formátu Base64 (HTTP).
WebAPI-Multipart	Technologie WebAPI, pole přenášeno jako Multipart/form-data. MIME typ binary/octet-stream (HTTP).
basicHttpBinding-Streaming	Technologie WCF, basicHttpBinding, TransferMode nastaveno na Streaming (HTTP).
basicHttpBinding-StreamingMTOM	Technologie WCF, basicHttpBinding, TransferMode nastaveno na Streaming, pro přenos binárních dat použit MTOM (HTTP).
basicHttpBinding-Buffering	Technologie WCF, basicHttpBinding, TransferMode nastaveno na Buffering (HTTP).
netTcpBinding-Streaming	Technologie WCF, netTcpBinding, TransferMode nastaveno na Streaming (Binární TCP).
netTcpBinding-Buffering	Technologie WCF, netTcpBinding, TransferMode nastaveno na Buffering (Binární TCP).

Tab. 8: Seznam způsobů přenosu dat pro směr od klienta - veřejný test

10 TESTOVACÍ APLIKACE

Popis níže je věnován laboratorní verzi. Změny veřejné verze oproti laboratorní jsou diskutovány na konci kapitoly.

10.1 Serverová část

Serverová část je realizována na operačním systému Microsoft Windows Server 2012 Datacenter Edition (MSDNAA licence), který je hostován ve virtuálním počítači v prostředí VirtualBox verze 4.2.6, s přidělenou RAM o velikosti 3 GiB a síťovou kartou v režimu „Síťový most“ s přímým přístupem do LAN, ke které je připojen hostující počítač. Hostující počítač je vybaven CPU Intel Core i3 na frekvenci 2,4 GHz a rotačním pevným diskem s rychlostí otáčení 7200 ot/min.

Zde se rotační pevný disk může jevit jako úzké hrdlo, testy ovšem vesměs probíhaly z RAM do RAM, tudíž vliv HDD lze zanedbat.

10.1.1 WebAPI

WebAPI část běží v rámci IIS verze 8.0. Jedná se o jednoduchou aplikaci. Implementace jednotlivým testů je naznačena níže.

Test serializace objektu:

```
[HttpGet]
[ActionName("Return5")]
public ReturnFormat5 Return5()
{
    return new ReturnFormat5()
    {
        Data1 = "-",
        Data2 = "A",
        Data3 = "B",
        Data4 = "C",
        Data5 = "D"
    };
}
```

Funkce vrací instanci objektu „ReturnFormat5“, který obsahuje 5 proměnných.

Test stahování binárních dat:

```
[HttpGet]
[ActionName("ReturnBinaryData")]
public ReturnBinary ReturnBinaryAuto(int bytes)
{
    byte[] data = GetBinaryData(bytes);
    return new ReturnBinary() { BinaryData = data };
}
```

Funkce vrací instanci objektu „ReturnBinary“ s vnitřním polem s náhodnými daty o dané délce.

Test uploadu binárních dat jako parametr metody

```
[HttpPost]
[ActionName("UploadFileByParameter")]
public ReturnInt UploadFileByParameter([FromBody]byte[] file)
{
    return new ReturnInt() { Integer = file.Length };
}
```

Funkce vrací instanci objektu „ReturnInt“, který obsahuje délku přijatých dat.

Test uploadu binárních dat jako Multipart/form-data

```
[HttpPost]
[ActionName("UploadFileByForm")]
public int UploadFileByForm()
{
    if (System.Web.HttpContext.Current.Request.Files.Count == 1)
    {
        foreach (string file in System.Web.HttpContext.Current.Request.Files)
        {
            var postedFile =
System.Web.HttpContext.Current.Request.Files[file];
            using (MemoryStream ms = new MemoryStream())
            {
                postedFile.InputStream.CopyTo(ms);
                byte[] fileBytes = ms.ToArray();
                return fileBytes.Length;
            }
        }
    }
}
```

```
        }  
    }  
    return 0;  
}
```

Funkce vrací délku přijaté zprávy.

Strukturu jednotlivých datových tříd lze snadno odvodit z jejich použití ve funkcích. Plný kód implementace WebAPI je uveden v datových přílohách k této práci.

10.1.2 WCF

WCF je implementována v self-hosted prostředí v rámci konzolové aplikace. Důvodem bylo zařazení udpBindingu do způsobů přenášení dat. Tento binding nelze použít při hostování v IIS verze 8.0.

Test serializace objektu:

```
public ReturnFormat5 Return5()  
{  
    return new ReturnFormat5()  
    {  
        Data1 = "-",  
        Data2 = "A",  
        Data3 = "B",  
        Data4 = "C",  
        Data5 = "D"  
    };  
}
```

Funkce vrací instanci objektu „ReturnFormat5“, který obsahuje 5 proměnných.

Test stahování binárních dat:

```
public ReturnBinary ReturnBinaryData(int bytes)  
{  
    byte[] data = GetBinaryData(bytes);  
    return new ReturnBinary() { BinaryData = data };  
}
```

Funkce vrací instanci objektu „ReturnBinary“ s vnitřním polem s náhodnými daty o dané délce.

Test uploadu binárních dat

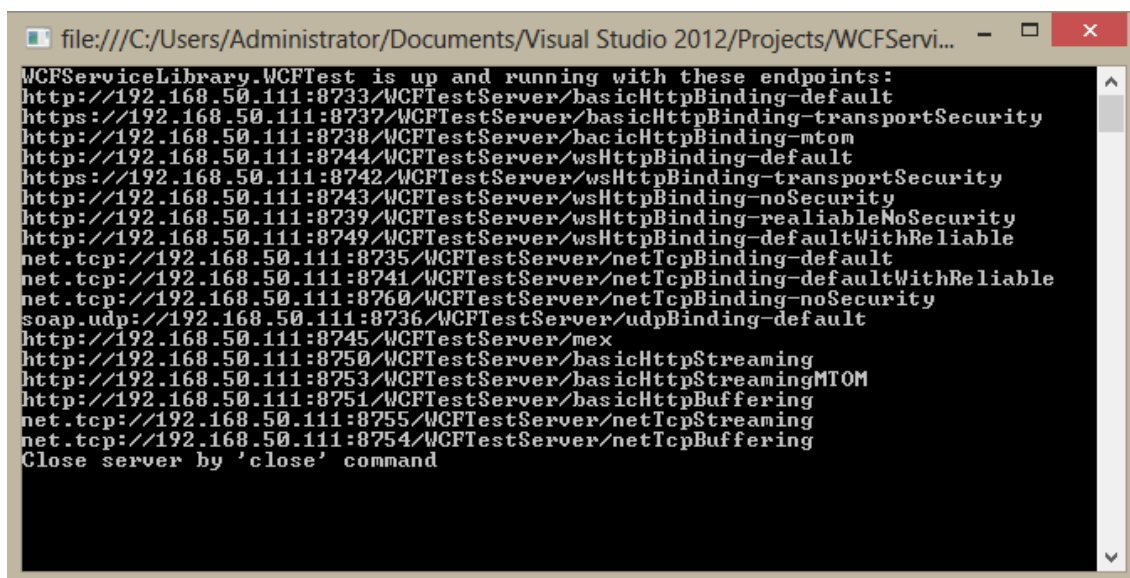
```
public UploadFileResponse UploadFile(UploadFileRequest file)
{
    System.IO.Stream uploadStream = file.FileByteStream;
    byte[] receivedBytes = new byte[0];

    byte[] buffer = new byte[32768];
    using (System.IO.MemoryStream ms = new System.IO.MemoryStream())
    {
        int read;
        while ((read = uploadStream.Read(buffer, 0, buffer.Length)) > 0)
        {
            ms.Write(buffer, 0, read);
        }
        receivedBytes = ms.ToArray();
    }
    uploadStream.Close();
    return new UploadFileResponse() { Size = receivedBytes.Length };
}
```

Funkce přebírá (referenci na) instanci objektu, který je serializovaný pomocí [MessageContract] a bere z ní (referenci na) instanci typu Stream. Následně přečte ze streamu všechny byty a uloží je do pole. Funkce vrací instanci objektu „UploadFileResponse“, který obsahuje proměnnou s délkou přijatých dat.

Je možno si všimnout, že metody týkající se složitosti objektu a stahování binárních dat jsou v obou frameworkcích (WebAPI i WCF) implementovány stejně. Rozdíl je v metodách pro uploadování binárních dat, protože se zde využívá specifických možností, které jsou dány použitým frameworkem.

Dané metody jsou zpřístupněny přes endpointy s bindingy, které jsou uvedeny v tabulce *Tab. 2* a *Tab. 4*, pro každý směr je vytvořen jiný kontrakt, a metody na stahování dat používají pouze endpointy z *Tab. 2* a metoda pro upload dat používá pouze endpointy uvedené v *Tab. 4*.



```
file:///C:/Users/Administrator/Documents/Visual Studio 2012/Projects/WCFServi...
WCFServiceLibrary.WCFTest is up and running with these endpoints:
http://192.168.50.111:8733/WCFTestServer/basicHttpBinding-default
https://192.168.50.111:8737/WCFTestServer/basicHttpBinding-transportSecurity
http://192.168.50.111:8738/WCFTestServer/basicHttpBinding-mtom
https://192.168.50.111:8742/WCFTestServer/wsHttpBinding-transportSecurity
http://192.168.50.111:8743/WCFTestServer/wsHttpBinding-noSecurity
http://192.168.50.111:8739/WCFTestServer/wsHttpBinding-reliableNoSecurity
http://192.168.50.111:8749/WCFTestServer/wsHttpBinding-defaultWithReliable
net.tcp://192.168.50.111:8735/WCFTestServer/netTcpBinding-default
net.tcp://192.168.50.111:8741/WCFTestServer/netTcpBinding-defaultWithReliable
net.tcp://192.168.50.111:8760/WCFTestServer/netTcpBinding-noSecurity
soap.udp://192.168.50.111:8736/WCFTestServer/udpBinding-default
http://192.168.50.111:8745/WCFTestServer/mex
http://192.168.50.111:8750/WCFTestServer/basicHttpStreaming
http://192.168.50.111:8753/WCFTestServer/basicHttpStreamingMTOM
http://192.168.50.111:8751/WCFTestServer/basicHttpBuffering
net.tcp://192.168.50.111:8755/WCFTestServer/netTcpStreaming
net.tcp://192.168.50.111:8754/WCFTestServer/netTcpBuffering
Close server by 'close' command
```

Obr. 9: Okno self-hostované WCF služby

Kompletní kód WCF služby je uveden v datových přílohách v této práci.

10.2 Klientská část

Testovací klient je vytvořen v prostředí .NET 4.5 v technologii WinForms. GUI umožňuje nastavit základní parametry testu a sledovat jeho průběh, samotné testy běží ve vlastním vlákne součásti Background Worker. Po dokončení testu je programem vygenerován report ve formátu „.txt“, který obsahuje výsledky jednotlivých měření.

Princip testu spočívá v provedení všech daných přenosů dat přes všechny dané protokoly daným počtem opakování. Před každým zavoláním funkce realizující přenos se sejme a uloží systémový čas, operace se spustí synchronně a dojde tedy k zablokování aktuálního vlákna, potom co se operace dokončí, se znovu sejme systémový čas a dané dva časy se od sebe odečtou. V případě, že přijatá data odpovídají očekávaným, tj. server vrátil buď přesný formát, nebo přesné množství očekávaných dat, a nebo stejnou délku jím přijatých dat, jako byla délka dat odeslaná klientem, je čas trvání zablokování vlákna s identifikací měřeného množství a daným způsobem přenosu zaznamenán. Na konci všech testů jsou data spárovány a vyexportovány v podobě reportu.

Při navrhování pořadí testování bylo nutno vyzkoušet několik různých způsobů volání jednotlivých metod pro přenos dat tak, aby byly výsledky co nejobektivnější.

U technologie WCF totiž často docházelo k postupnému zrychlování, kdy první provedení metody trvalo nejdéle, druhé již trvalo kratší dobu a od třetího provedení metody byla doba provádění konstantní. Tento jev je možno vysvětlit pravděpodobnou optimalizací, kterou WCF provádí. Daný jev se projevoval jak u volání těžší metody, tak i u volání jiných metod přes daný kanál. Časová prodleva prvního volání se objevila i u WebAPI. U obou technologií lze zdržení prvního volání vysvětlit vybudováním TCP spojení se serverem (cca 2s) a následným ustanovováním rychlosti přenosu (změna velikosti TCP okna). Samotné pořadí volání jednotlivých metod je tedy u každého způsobu zvoleno jinak, tak, aby se dosáhlo nejmenšího zkrácení. Tyto pořadí lze vyčíst z příložených zdrojových kódů v datových přílohách v této práci.

Ukázky kódu testovacího programu:

Volání metody přes WebAPI:

```
requestTime = DateTime.Now;
response = client.GetAsync("api/Test/Return5").Result;
responseTime = DateTime.Now;
response.EnsureSuccessStatusCode();
if (response.IsSuccessStatusCode)
{
    ReturnFormat5 products =
    response.Content.ReadAsAsync<ReturnFormat5>().Result;
    TimeSpan timespan = responseTime - requestTime;
    _measuredTimes.Add(new MeasuredTime() { MeasureType = "WebAPI-Format5-
JSON", Milisecons = timespan.TotalMilliseconds });
}
```

Na ukázce je jedno měření při použití technologie WebAPI.

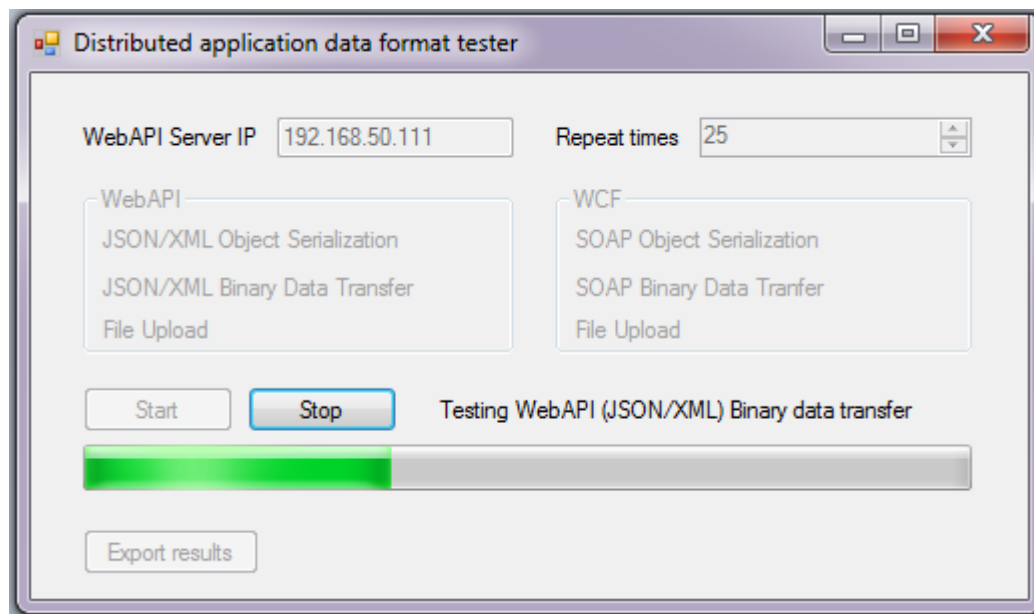
Implementace měření stahování binárních dat:

```
for (int i = 0; i < bindings.Count(); i++)
{
    bindings[i].client.Open();
    bindings[i].client.InnerChannel.OperationTimeout = new TimeSpan(1, 0, 0);
    bindings[i].client.ReturnBinaryData(100);
    foreach (int size in dataSiz)
    {
        if (bindings[i].name == "udp-default" && size > 10000) continue;
```

```
for (int i1 = 0; i1 < _repeatTimes; i1++)
{
    requestTime = DateTime.Now;
    var data = bindings[i].client.ReturnBinaryData(size);
    responseTime = DateTime.Now;
    if (data.BinaryData.Count() == size)
    {
        TimeSpan timespan = responseTime - requestTime;
        _measuredTimes.Add(new MeasuredTime() { MeasureType = "WCF-
Binary-" + size + "-" + bindings[i].name, Miliseconds = timespan.TotalMilliseconds
});
    }
}
bindings[i].client.Close();
}
```

Na ukázce je soustava měření stahování binárních dat přes technologii WCF. Je zde možno vidět integraci měření jedné hodnoty do celku. Při použití udpBindingu je měření dat větších jak 10 kB přeskočeno. Bohužel se mi nepodařilo nijak na tomto bindingu zvýšit omezení dat nad výchozích 64 kB a při velikosti binárních dat 50 kB (které se po zakódování do base64 a vložení do zprávy zvětší nad 64 kB) nedocházelo k žádné odpovědi. Přestože se může zdát, že návrh vnějších a vnitřních smyček není optimální, opak je pravdou, původně po každé jedno opakování proběhlo měření všech hodnot přes všechny bindingy (což mělo eliminovat výkyvy rychlosti a odezvy sítě v čase a poskytnout méně zkreslené hodnoty). Praktické měření ovšem ukázalo, že pokud jsou vnitřní a vnější smyčky navrženy tak, jak je uvedeno na úryvku kódu výše, jsou data zkreslena méně, než kdyby byl počet měření implementován ve vnějších smyčkách.

Celý kód služby včetně nastavení jednotlivých bindingů je možno nalézt v datové příloze k této práci.



Obr. 10: Testovací klient - laboratorní měření

10.3 Změna u veřejného testování

Softwarová podpora veřejného testování vychází z původní aplikace pro testování laboratorní, došlo pouze ke kosmetickým změnám.

10.3.1 Serverová část

Jako server pro veřejné testování byl zvolen virtuální server hosting u firmy Wedos Internet, a.s. s operačním systémem Microsoft Windows Server 2012, běžící na sdíleném jádře o frekvenci 1,8 Ghz a mající k dispozici 1 GiB RAM. Tyto hodnoty jsou naprostým minimem pro běh tohoto operačního systému a do určité míry zkreslují naměřené výsledky, z celkového hlediska naměřených hodnot to ale není významné. Důvodem volby takto „slabé“ konfigurace je cena provozu. Server je opatřen důvěryhodným SSL certifikátem vystaveným společností Comodo CA Limited (zdarma s platností 90 dní), který umožňuje testovat přenos dat přes protokol HTTPS.

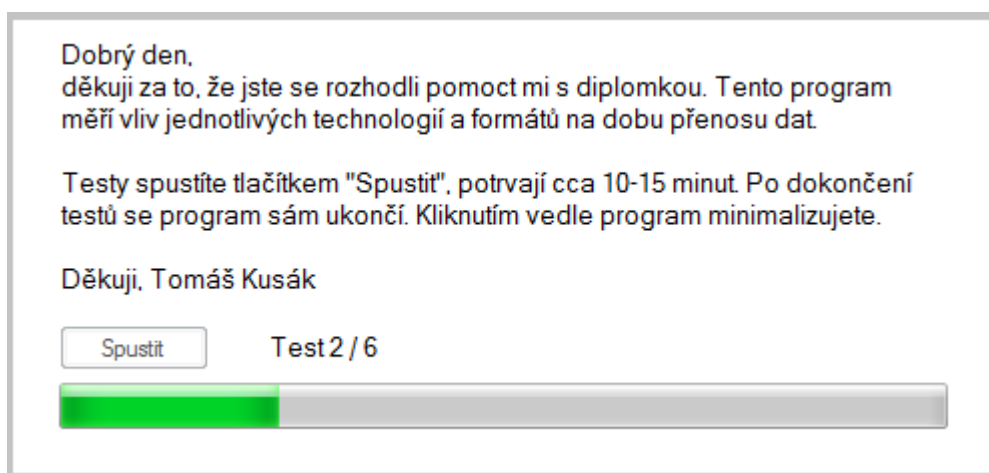
Do WebAPI části je přidána metoda umožňující přijmout naměřená data z klienta a uložit je na server, WCF část byla přesunuta na jinou doménu. Server je přístupný přes doménu tkdata.eu.

10.3.2 Klientská část

Klientská část podstoupila oproti laboratorní několik změn:

- Byl upraven rozsah měřených dat dle *Tab. 5* a *Tab. 7* a odstraněn udpBinding.
- Aplikace byla rozšířena o zachytávání výjimek (v laboratorním prostředí je žádoucí, aby aplikace při chybě spadla, v reálném prostředí to, aby o výjimce uživatel vůbec nevěděl).
- Aplikace si sama změří rychlost internetu a na základě toho nastaví počet opakování testů tak, aby celková doba měření nepřesáhla 15 minut. Na každý 1Mbit/s rychlosti ke klientovi se provede jedno opakování stahování dat a 1/3 testu uploadování dat, hodnota je vždy zaokrouhlena nahoru (tj. 1/3 -> 1).
- Výsledky měření nejsou zobrazeny uživateli, místo toho jsou odeslány na server.
- Aplikace má upraveno GUI tak, aby ji mohl ovládat opravdu každý. Po spuštění aplikace je jedinou možností provést test tlačítkem „Spustit“, aplikace se sama ukončí po provedení testu. Zamezí se tak nechtěnému ukončení aplikace v průběhu testování.

Na server se ukládají pouze výsledky testu a datum a čas dokončení, žádné údaje identifikující uživatele nebo měřící počítač se neukládají.



Obr. 11: Testovací klient – veřejné měření

11 NAMĚŘENÉ HODNOTY

Tabulky níže obsahují naměřené hodnoty. Výsledné hodnoty jsou z naměřených dat získány dvěma způsoby:

- Aritmetický průměr – zohledňuje krátkodobé výkyvy služby, hodnota je zkreslená „ustřelenými“ hodnotami, ale v kontextu celkového počtu měření přináší věrnější pohled na chování služby jako celku z hlediska delšího časového horizontu (kdy se „ustřelené“ hodnoty v určitém časovém rámci opakují)
- Medián – eliminuje krátkodobé výkyvy, z hlediska časového chování služby a naměřených dat určuje hodnotu, s jakou služba reagovala nejčastěji, dává reálnější pohled na přenos samotných dat, nezohledňuje celkové chování služby v delším časovém horizontu

11.1 Laboratorní měření

Podkapitoly níže obsahují zpracované výsledky ve formátu průměr / medián pro jednotlivé testy. Každé jednotlivé měření bylo opakováno 50x. Toto číslo se může zdát nízké, ale naměřené hodnoty byly pro 50 opakování (a protokol TCP) překvapivě téměř shodné, takže reálný rozdíl v počtu opakování měření na samotný výsledek (průměr/medián) je zanedbatelný a dané statistické charakteristiky by vyšly téměř shodné pro jakýkoliv počet měření vyšší než 10x.

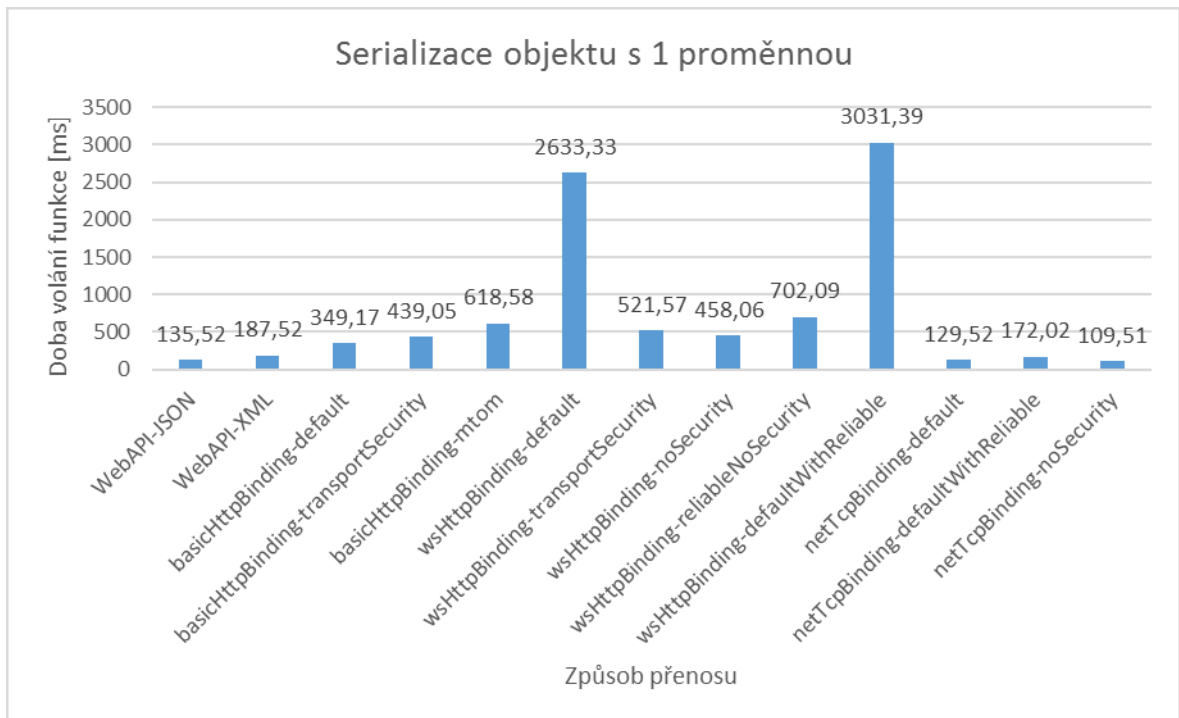
Pro grafické znázorňování výsledků měření jsou použity vypočtené hodnoty mediánu.

11.1.1 Výsledky testu přenášení serializovaného objektu

Objekt s různým počtem proměnných byl stahován ze serveru na klienta (viz. Tab. 1, datové jednotky Format1-Format40) způsoby uvedenými v Tab. 2, byl měřen čas přenosu v ms a statisticky zpracován. Jeden nejlepší a nejhorší výsledek z každého měření byl zahozen.

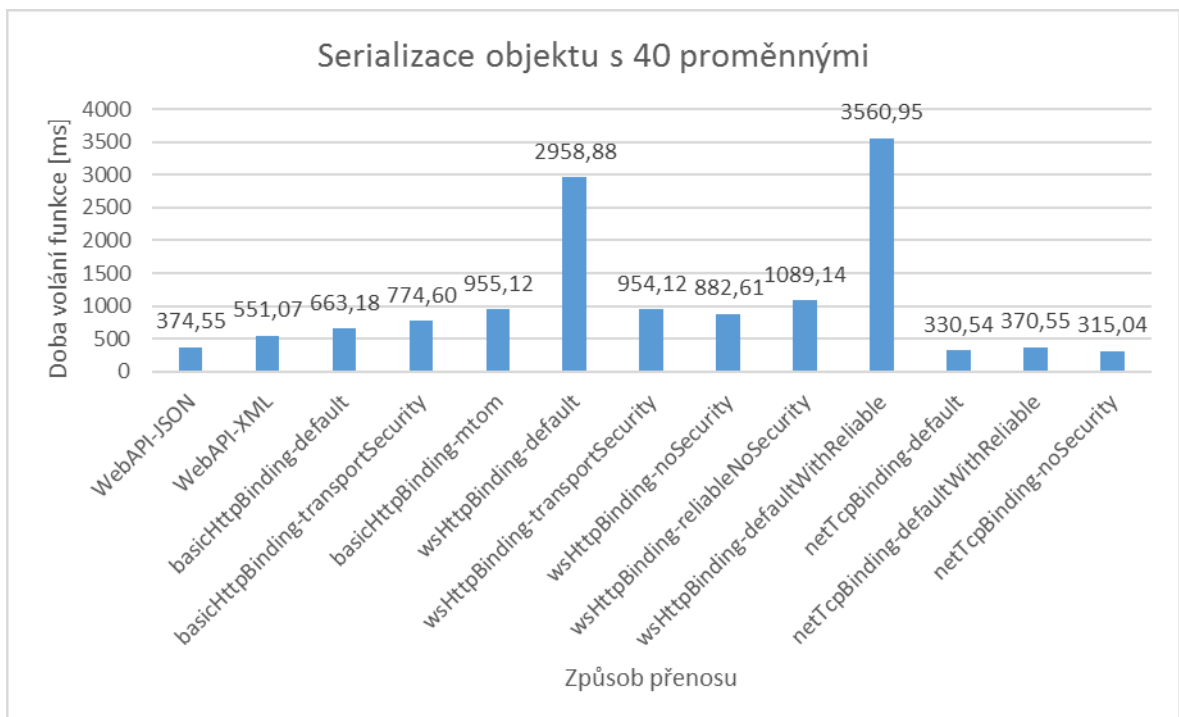
Průměr [ms]	Format 1	Format 5	Format 10	Format 20	Format 40
Medián [ms]					
Rozptyl [-]					
WebAPI-JSON	135,46	156,54	184,12	248,71	376,03
	135,52	155,52	185,02	248,53	374,55
	0,07	18,59	17,82	5,04	92,45
WebAPI-XML	187,28	238,65	280,58	370,87	550,99
	187,52	238,53	280,54	370,55	551,07
	81,76	0,43	0,20	1,22	1,29
basicHttpBinding- default	350,31	370,76	420,15	516,13	664,52
	349,17	372,10	419,31	516,28	663,18
	9,19	1,12	0,37	3033,79	6100,22
basicHttpBinding- transportSecurity	440,21	458,12	514,07	617,98	774,70
	439,05	458,06	514,07	618,08	774,60
	32,69	0,40	0,50	0,43	0,37
basicHttpBinding- mtom	618,60	622,37	645,26	742,17	949,46
	618,58	621,01	644,58	742,59	955,12
	0,35	6,73	11,17	5,42	152,65
wsHttpBinding- default	2633,57	2653,44	2726,93	2814,64	2960,80
	2633,33	2655,84	2726,35	2815,36	2958,88
	25,87	66,88	1,02	15,56	59,19
wsHttpBinding- transportSecurity	520,47	562,11	618,26	729,33	954,80
	521,57	562,07	618,08	729,59	954,12
	31,23	0,27	1,39	1,52	12,44
wsHttpBinding- noSecurity	457,41	498,54	552,03	663,72	881,71
	458,06	498,56	552,07	662,58	882,61
	11,70	0,09	0,04	7,79	7,51
wsHttpBinding- reliableNoSecurity	702,15	742,31	743,39	995,53	1089,28
	702,09	742,09	742,59	995,63	1089,14
	0,11	0,71	16,62	20,20	10,14
wsHttpBinding- defaultWithReliable	3031,04	3071,64	3124,67	3235,33	3561,66
	3031,39	3071,14	3124,90	3234,91	3560,95
	6,58	10,48	15,32	29,85	22,45
netTcpBinding- default	129,48	186,58	202,39	250,51	330,50
	129,52	186,52	202,53	250,53	330,54
	0,24	0,13	0,11	1,61	1,18
netTcpBinding- defaultWithReliable	171,77	229,03	240,03	289,23	370,66
	172,02	228,53	242,53	290,04	370,55
	0,17	1,25	4,24	2,19	0,34
netTcpBinding- noSecurity	109,26	169,02	186,54	231,99	314,85
	109,51	169,02	186,52	231,53	315,04
	0,40	0,13	0,08	5,82	8,08

Tab. 9: Serializace objektu – stahování 16 kbit/s



Graf 1: Serializace objektu s 1 proměnnou - 16 kbit/s

Na grafu výše jsou demonstrovány časy nutné k přenosu objektu s jednou proměnnou. Graf lze orientačně použít pro určení režie samotných protokolů.



Graf 2: Serializace objektu s 40 proměnnými - 16 kbit/s

11.1.2 Výsledky testu stahování binárních dat

Objekt s jedním polem o proměnné délce byl stahován ze serveru na klienta rychlostí 16 kbit / s (viz. Tab. 1, datové jednotky Binary1-Binary1k) způsoby uvedenými v Tab. 2, byl měřen čas přenosu v ms a statisticky zpracován. Jeden nejlepší a nejhorší výsledek z každého měření byl zahozen.

Průměr [ms]	Binary 1	Binary 5	Binary 10	Binary 50	Binary 100	Binary 500	Binary 1k
Medián [ms]							
Rozptyl [-]							
WebAPI-JSON	245,71	138,31	142,81	168,98	203,57	469,54	727,57
	215,53	139,52	143,02	169,02	203,53	469,56	727,59
	118,34	19,71	0,45	0,63	0,61	0,51	1,32
WebAPI-XML	286,26	213,43	217,45	244,17	278,18	544,03	879,27
	286,54	214,03	218,03	244,03	278,04	544,07	878,61
	0,66	2,29	3,29	0,09	0,27	0,40	10,32
basicHttpBinding- default	390,51	378,01	381,95	407,99	442,00	708,39	1042,49
	376,05	378,05	382,05	408,05	442,06	708,59	1042,63
	5063,05	0,06	5,02	0,13	0,09	0,38	0,39
basicHttpBinding- transportSecurity	460,44	442,02	450,00	474,00	505,98	777,90	1106,10
	442,06	442,06	450,06	474,06	506,06	778,10	1106,14
	102,19	0,56	0,13	9,01	0,05	0,71	5,90
basicHttpBinding- mtom	601,18	577,97	582,01	608,04	641,94	908,46	1225,34
	576,07	578,07	582,07	608,08	642,08	908,62	1225,16
	831,97	0,45	4,53	31,08	8,55	2,84	2,54
wsHttpBinding- default	2663,40	2666,56	2669,34	2695,72	2707,22	2902,87	3326,44
	2664,34	2666,34	2669,34	2696,34	2706,34	2898,87	3330,42
	44,28	1,87	9,18	12,76	51,69	201,95	369,30
wsHttpBinding- transportSecurity	553,95	553,99	561,97	585,99	618,02	890,01	1217,91
	554,07	554,07	562,07	586,07	618,08	890,11	1218,15
	121,26	5,20	1,27	6,72	17,56	32,10	5,83
wsHttpBinding- noSecurity	509,16	489,50	493,44	519,61	553,35	816,88	1157,15
	487,56	489,56	493,56	519,57	553,57	820,10	1154,15
	1741,43	6,39	16,15	14,04	5,36	5270,24	252,36
wsHttpBinding- reliableNoSecurity	745,11	734,01	737,99	763,18	797,62	1274,24	1398,96
	732,09	734,09	738,09	764,10	799,10	1065,14	1398,68
	2718,10	18,79	7,75	25,22	49,35	5587,64	0,10
wsHttpBinding- defaultWithReliable	3099,39	3104,93	3086,23	3113,18	3149,60	3459,34	3750,40
	3079,89	3082,39	3085,89	3112,90	3148,40	3460,44	3747,48
	812,70	3235,27	6,00	7,22	11,81	104,96	85,06
netTcpBinding- default	142,56	129,96	130,50	154,46	178,48	393,63	651,54
	130,52	130,52	130,52	154,52	178,52	378,55	626,58
	13,38	0,92	3,87	3,69	15,33	544,49	5055,01
netTcpBinding- defaultWithReliabl	169,46	170,46	170,48	194,50	218,53	416,55	876,91
	170,52	170,52	170,52	194,52	218,53	416,55	666,58
	19,71	0,11	0,10	0,33	1,04	47,53	7658,89
netTcpBinding- noSecurity	110,03	112,95	115,51	135,50	160,48	375,79	635,10
	110,01	113,01	115,51	135,52	160,52	361,05	611,08
	0,03	0,23	0,10	17,95	0,12	499,81	1364,84

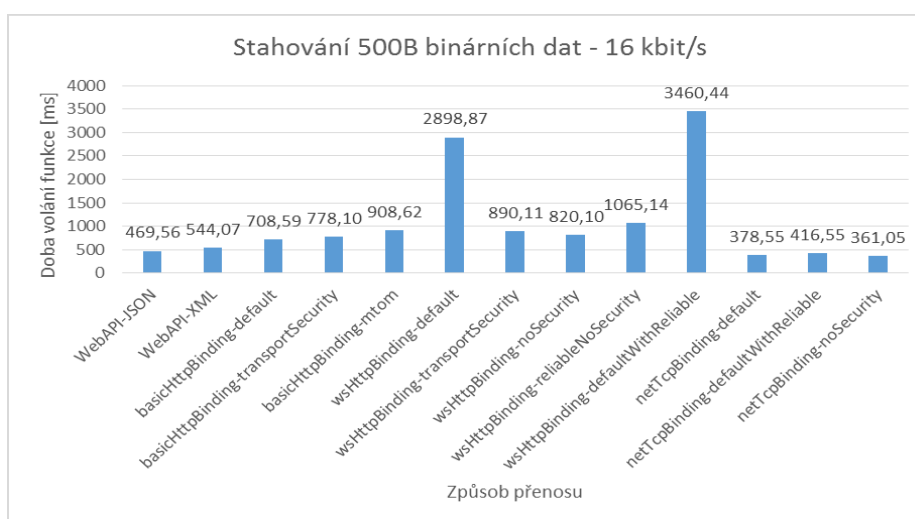
Tab. 10: Stahování binárních dat – část 1/3 – 16 kbit/s

V tabulce se často objevuje jev, kdy přenos 1B dat trvá déle než přenos 5B. Toto je pravděpodobně způsobeno metodikou testu, v důsledku níž v průběhu testu Binary1 probíhala dohoda na velikosti okna protokolu TCP a data tedy nebyla přenášena plnou možnou rychlostí. Od přenosu dat o velikosti 50B a výš lze výsledky považovat za relevantní. Z tabulky lze přehledně vyčíst, že časový rozdíl mezi přenosem 2 rozdílných velikostí například 50B a 100B (přibližně 30-40 ms, kromě UDP) je u všech protokolů řádově stejný a je zanedbatelný vůči základovému posunu (například 130 ms u netTcpBindingu a 2650 ms u wsHttpBindingu). Právě základový posun hodnot o počáteční hodnotu prozrazuje hodně o efektivitě a rychlosti jednotlivých protokolů.

Z hodnot (pro data o velikostech 1B – 1KB) lze dále vyčíst tyto rozdíly:

- Přenos přes HTTPS trvá o přibližně o 60ms déle než přes HTTP (Transport Security).
- Zabezpečení na úrovni zprávy protáhne dobu přenosu přibližně o 2000 ms (Message Security).
- Garantované doručení zpráv v pořadí, v jakém byly odeslány, protáhne přenos přibližně o 0-400 ms v závislosti na bindingu.

Z časových hodnot základového posuvu a jednotlivých poznatků a znalosti rychlosti komunikace 2 kB/s lze přibližně odvodit velikost režie jednotlivých protokolů a vlastností. Je ovšem možno použít přesnější metodu a vyjít přímo ze zachycených dat v průběhu komunikace.



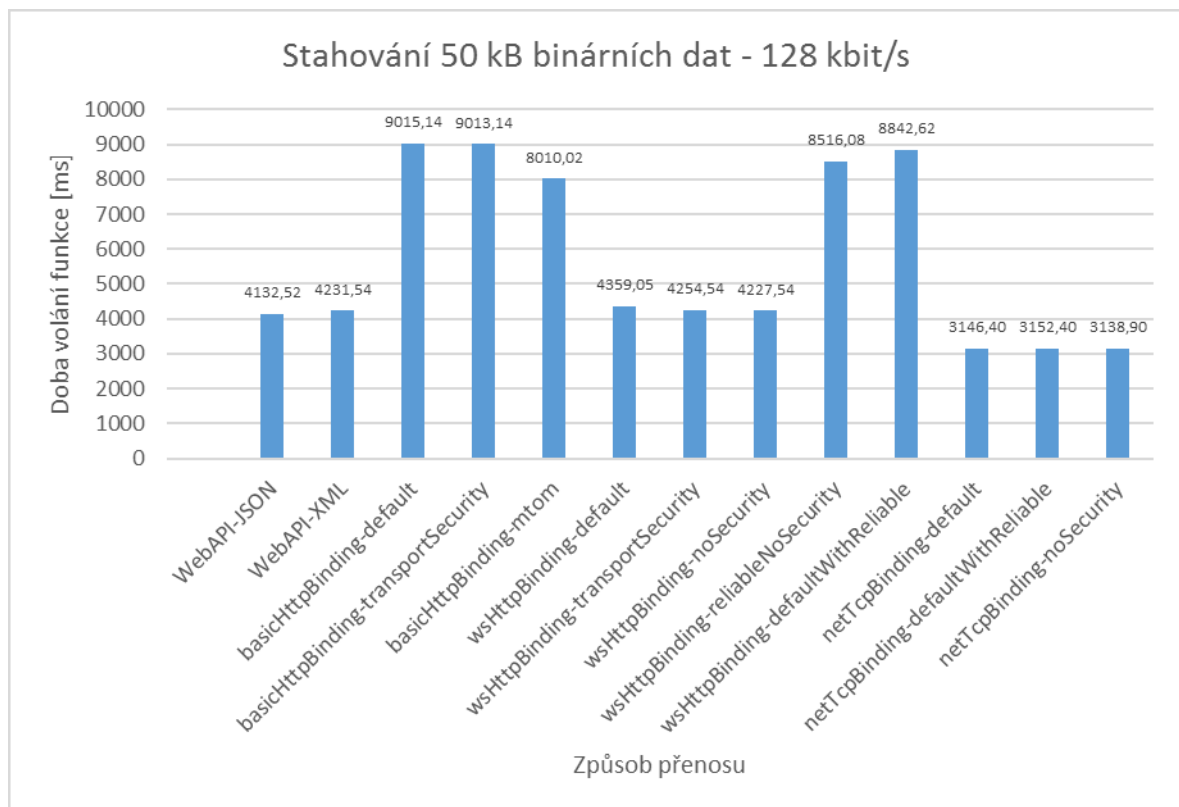
Graf 3: Stahování 500B binárních dat – 16 kbit/s (vybraná hodnota)

V následující tabulce jsou uvedeny časy stahování objektu s jedním polem o proměnné délce, objekt byl stahován ze serveru na klienta rychlostí 128 kbit / s (viz. *Tab. 1*, datové jednotky Binary5k-Binary100k) způsoby uvedenými v *Tab. 2*, byl měřen čas přenosu v ms a statisticky zpracován. Jeden nejlepší a nejhorší výsledek z každého měření byl zahozen.

Průměr [ms]	Binary 5k	Binary 10k	Binary 50k	Binary 100k
Medián [ms]				
Rozptyl [-]				
WebAPI-JSON	372,01	876,79	4132,92	8374,44
	374,55	887,11	4132,52	8375,06
	187,53	436,52	13,26	18,07
WebAPI-XML	405,73	895,77	4231,12	8302,45
	403,55	898,61	4231,54	8302,55
	250,18	349,78	100,48	63,16
basicHttpBinding- default	453,78	878,29	8825,64	13121,51
	463,56	880,11	9015,14	13122,67
	129,07	126,33	2214,55	40,60
basicHttpBinding- transportSecurity	471,62	887,17	8826,88	13216,24
	472,06	889,11	9013,14	13215,18
	15,85	105,23	1211,09	20,82
basicHttpBinding- mtom	402,37	714,13	7700,53	11114,53
	403,05	715,59	8010,02	11115,41
	10,17	61,53	12105,17	31,89
wsHttpBinding- default	608,22	1027,53	4357,23	8527,30
	609,08	1025,63	4359,05	8526,08
	27,22	75,47	118,73	261,21
wsHttpBinding- transportSecurity	486,84	900,21	4255,70	8440,91
	486,56	902,11	4254,54	8440,57
	15,05	166,96	266,32	17,92
wsHttpBinding- noSecurity	478,28	892,19	4229,22	8395,51
	477,56	894,11	4227,54	8394,57
	38,85	118,88	76,23	38,52
wsHttpBinding- reliableNoSecurity	506,20	1041,01	8684,54	16619,31
	508,06	924,62	8516,08	16849,64
	72,41	9419,06	653,05	21563,93
wsHttpBinding- defaultWithReliable	673,19	2124,21	8757,33	16977,30
	671,09	2176,28	8842,62	17176,18
	88,04	1666,07	4358,24	3788,65
netTcpBinding- default	323,98	643,36	3146,18	6275,52
	328,54	641,08	3146,40	6277,30
	422,37	103,05	3,24	67,40
netTcpBinding- defaultWithReliable	331,38	648,34	3152,22	6241,29
	333,04	646,08	3152,40	6241,29
	73,53	105,35	14,11	3108,56
netTcpBinding- noSecurity	323,51	641,61	3136,90	6262,87
	326,54	639,08	3138,90	6263,80
	111,46	105,57	54,48	18,40

Tab. 11: Stahování binárních dat – část 2/3 – 128 kbit/s

Z tabulky lze vyčíst „divné“ chování basicHttpBindingu, wsHttpBindingu s vypnutým zabezpečením a wsHttpBindingu s ReliableMessagingem, kdy se doba přenosu 50kB dat (na aplikační vrstvě) a víc neúměrně protáhla. Všimnout si lze rovněž MTOM optimalizace, která přenáší binární data přímo binárně (na rozdíl od jiných http přenosů, kde jdou zakódována do textu pomocí base64), binární přenos dat rovněž provádí netTcpBinding. Rychlejší přenos objektu přes JSON-XML je pravděpodobně způsoben chybou limitování rychlosti.



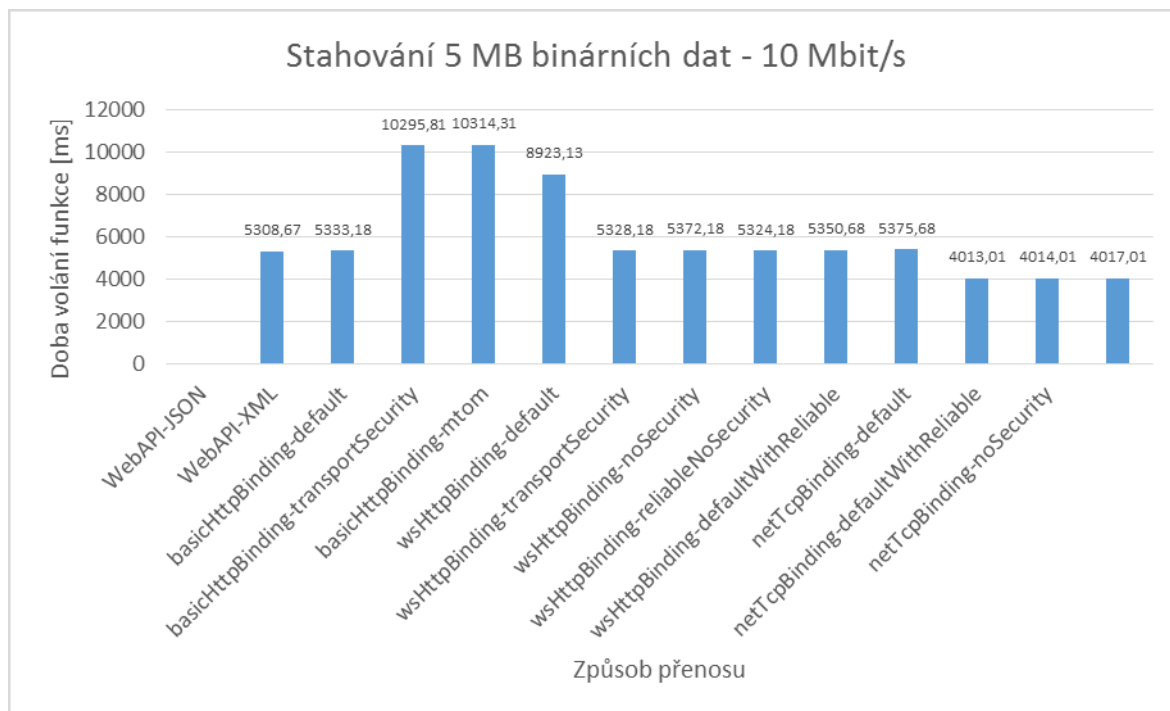
Graf 4: Stahování 50 kB binárních dat – 128 kbit/s (vybraná hodnota)

Další tabulka uvádí časy stahování objektu s jedním polem o proměnné délce, objekt byl stahován ze serveru na klienta rychlostí 10 Mbit / s (viz. Tab. 1, datové jednotky Binary500k-Binary10M) způsoby uvedenými v Tab. 2, byl měřen čas přenosu v ms a statisticky zpracován. Jeden nejlepší a nejhorší výsledek z každého měření byl zahozen.

Průměr [ms]	Binary 500k	Binary 1M	Binary 5M	Binary 10M
Medián [ms]				
Rozptyl [-]				
WebAPI-JSON	388,11	1059,03	5314,81	10667,85
	369,05	1033,63	5308,67	10638,35
	1709,22	2371,04	5634,68	13000,80
WebAPI-XML	282,38	1047,75	5364,54	10692,56
	280,54	1037,63	5333,18	10658,85
	82,48	717,27	10543,03	7766,19
basicHttpBinding- default	5138,67	5871,61	10304,45	15827,89
	5339,68	5868,25	10295,81	15789,51
	905,60	349,48	2523,59	14716,95
basicHttpBinding- transportSecurity	5181,74	5881,71	10333,09	15874,60
	5353,18	5879,75	10314,31	15836,01
	20684,84	434,77	2834,78	7939,57
basicHttpBinding- mtom	4972,07	5602,01	8929,29	13063,54
	5169,16	5602,21	8923,13	13042,66
	717,45	113,57	454,49	8650,98
wsHttpBinding- default	525,05	1079,60	5335,80	10710,78
	534,07	1065,64	5328,18	10698,36
	2523,87	2359,52	1475,66	1134,64
wsHttpBinding- transportSecurity	536,35	1072,36	5382,76	10800,99
	536,57	1054,63	5372,18	10768,37
	10342,47	1172,52	5175,61	7007,14
wsHttpBinding- noSecurity	525,57	1067,34	5344,80	10786,35
	529,57	1053,63	5324,18	10749,87
	399,69	1252,48	3335,13	12358,37
wsHttpBinding- reliableNoSecurity	523,61	1282,16	6402,51	12013,09
	528,07	1111,14	5350,68	10725,36
	456,31	17756,92	64997,91	7670,48
wsHttpBinding- defaultWithReliable	525,31	1201,11	7044,05	12427,30
	531,07	1067,14	5375,68	10745,36
	668,96	21028,44	42992,39	20347,21
netTcpBinding- default	390,01	803,42	4006,77	8013,58
	377,05	812,10	4013,01	8013,02
	2619,47	904,05	1043,12	838,09
netTcpBinding- defaultWithReliable	398,19	800,32	4010,09	8027,52
	372,55	806,10	4014,01	8027,52
	1614,17	800,48	7036,90	1311,12
netTcpBinding- noSecurity	399,35	801,26	3998,93	7999,52
	374,05	792,60	4017,01	8017,02
	1617,45	793,80	1524,46	1369,91

Tab. 12: Stahování binárních dat – část 3/3 – 10 Mbit/s

Naměřená data zde spíše odhalují rychlost frameworku při zpracování dat přenášených přes různé protokoly. Rozdíl je také patrný mezi daty přenášenými v textové formě (base64) a daty přenášenými nativně binárně.



Graf 5: Stahování 5 MB binárních dat – 10 Mbit/s (vybraná hodnota)

11.1.3 Výsledky testu nahrávání binárních dat

Pole bytů o dané velikosti bylo uploadováno z klienta na server (viz. Tab. 3, datové jednotky Upload500k-Upload10M) způsoby uvedenými v Tab. 4, byl měřen čas přenosu v ms a statisticky zpracován. Jeden nejlepší a nejhorší výsledek z každého měření byl zahozen.

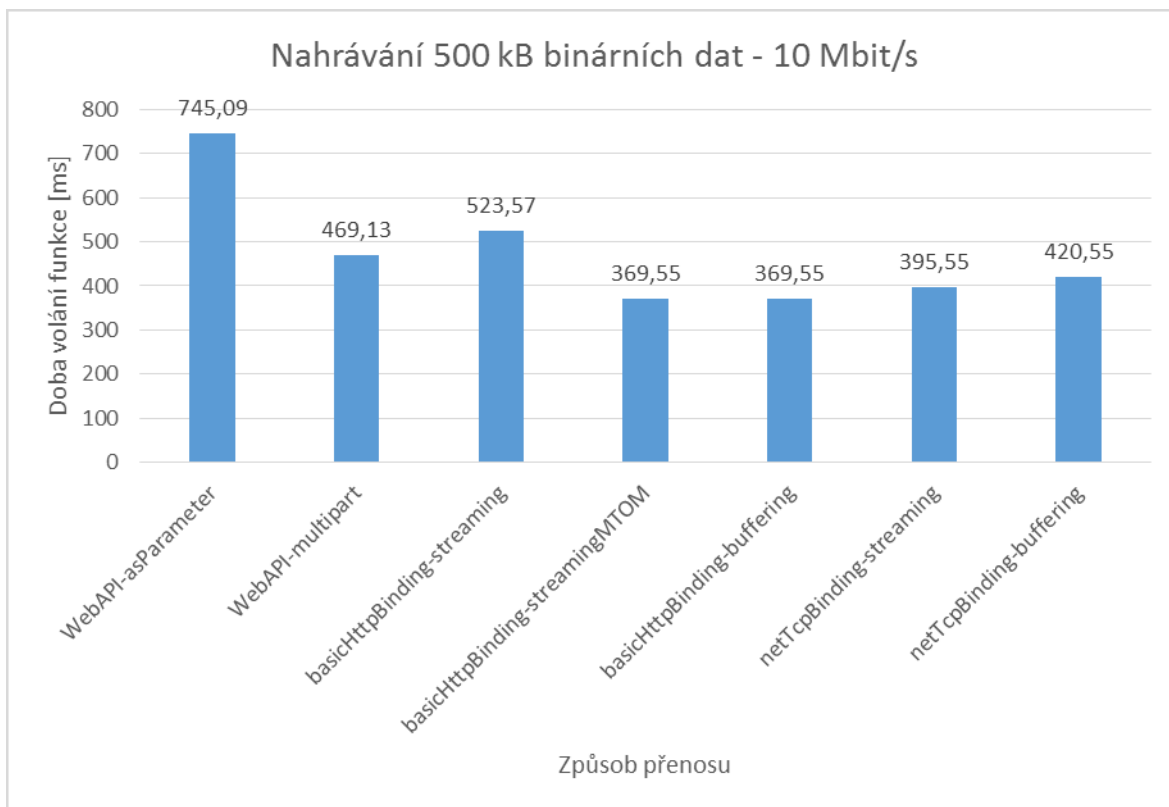
Průměr [ms]	Upload 500k	Upload 1M	Upload 5M	Upload 10M
Medián [ms]				
Rozptyl [-]				
WebAPI- asParameter	731,77	1602,82	9154,18	18498,79
	745,09	1591,20	9050,65	18398,34
	2253,49	1394,88	35121,59	519144,09
WebAPI- multipart	469,00	802,46	3950,12	7892,42
	469,13	807,10	3958,50	7888,00
	746,39	2163,72	1885,74	1183,15
basicHttpBinding- streaming	517,53	1047,97	5281,79	10573,18
	523,57	1045,63	5281,17	10577,34
	634,56	595,85	1547,72	1345,40
basicHttpBinding- streamingMTOM	386,17	786,40	3931,60	7862,74
	369,55	780,60	3936,50	7865,50
	1479,16	275,09	1546,09	1280,51
basicHttpBinding- buffering	522,73	1047,91	5252,05	10596,29
	520,57	1032,63	5242,17	10594,85
	543,95	922,34	1927,88	1464,16
netTcpBinding- streaming	379,73	779,50	3909,06	7814,93
	395,55	773,60	3913,00	7808,99
	1451,49	1456,17	1435,65	2526,85
netTcpBinding- buffering	388,67	779,76	3912,00	7816,35
	420,55	760,10	3897,50	7806,99
	1616,07	1260,33	2314,33	1937,70

Tab. 13: Nahrávání binárních dat – 10 Mbit/s

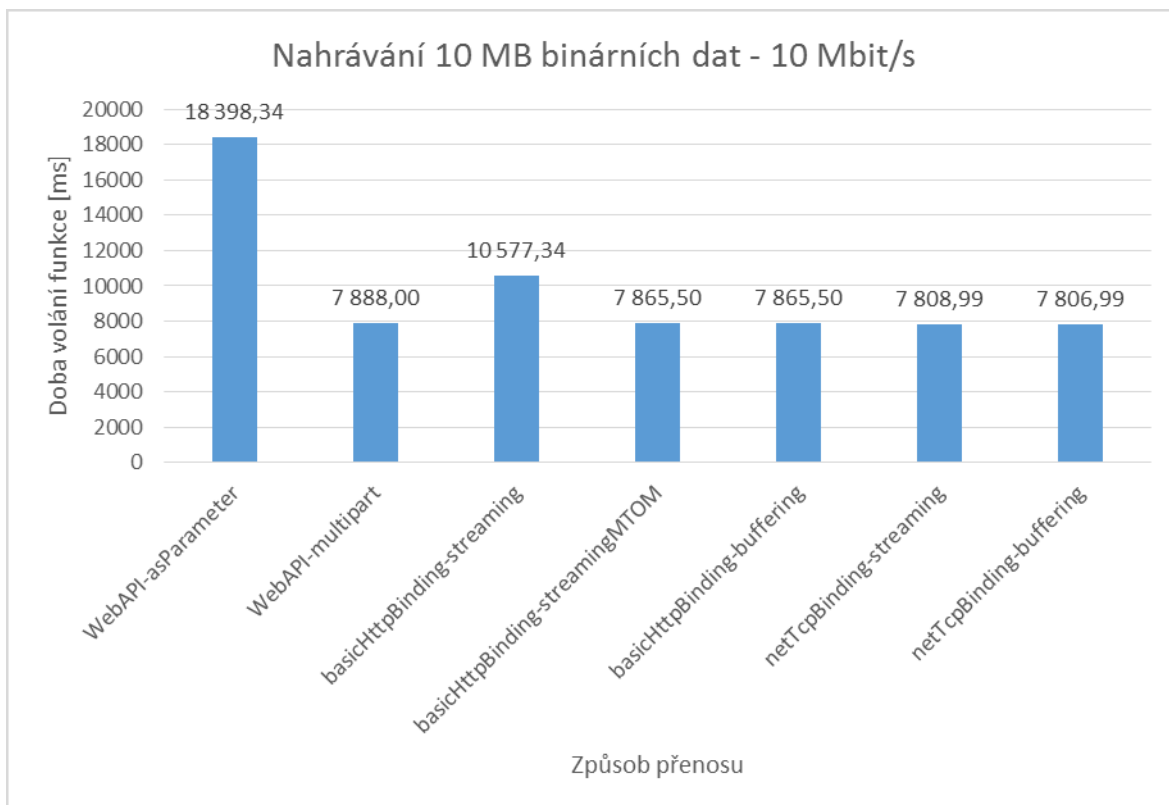
Režie jednotlivých protokolů je u těchto objemů dat zanedbatelná. Do popředí vystupuje základní rozdíl v přenosu a to textový (pole serializováno pomocí base64), kde je přibližná doba uploadu 10 MB dat přibližně 10500 ms, a binární, kde je přibližná doba uploadu téže dat přibližně 7800 ms. Rozdíl 7800 ms a 10500 ms přehledně ukazuje aditivní režii 33% při přenosu binárních dat v textové formě.

Rozdíl mezi zpracováním zprávy v cíli až po jejím celém nabufferování vs. v průběhu nahrávání není znatelný, data se totiž musela v obou případech nahrát na serveru do pole bytů, až byla nahrána úplně všechna, teprve potom server odpověděl klientovi.

WebAPI-asParameter uvádí, jak dlouho se pole bytů odesílalo v parametru metody, WebAPI-multipart simuluje klasické nahrávání souboru z webového prohlížeče. Přenos 500kB dat rychlostí 10 Mbit/s, tj. 1 250 kB/s, neumožňuje data přenést rychleji než za 400 ms), nižší hodnoty mohou být způsobeny krátkodobými špičkami a nepřesností při limitování maximální rychlosti. Rychlost je nicméně totožná pro všechny datové objemy, tudíž je test relevantní.



Graf 6: Nahrávání 500 kB binárních dat – 10 Mbit/s (vybraná hodnota)

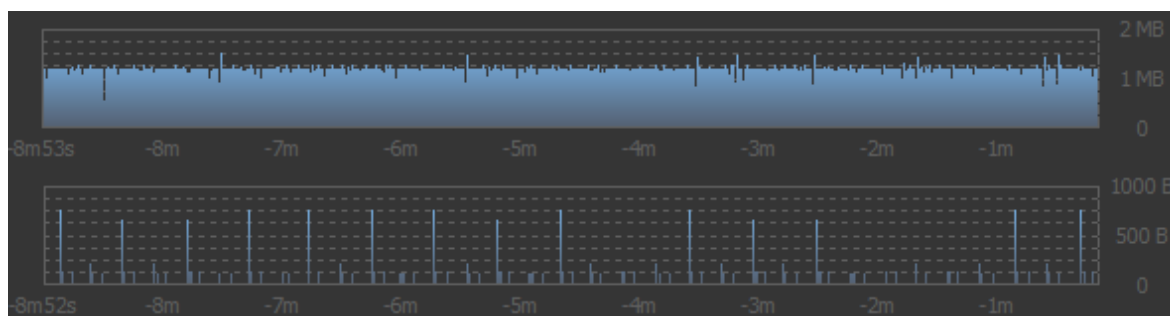


Graf 7: Nahrávání 10 MB binárních dat – 10 Mbit/s (vybraná hodnota)

11.1.4 Zaznamenané průběhy přenášení dat v čase

V průběhu testu byly pro jednotlivé způsoby přenosu zaznamenány typická chování v průběhu času. Vrchní obrázek stahování / dolní odesílání dat. Dolní část obsahuje dotazy na server, horní část odpovědi na ně.

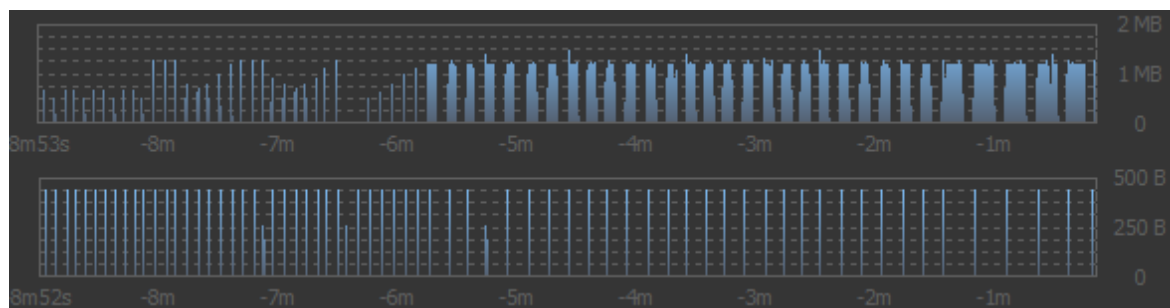
Stahování dat přes WebAPI



Obr. 12: Průběh stahování dat přes WebAPI v čase

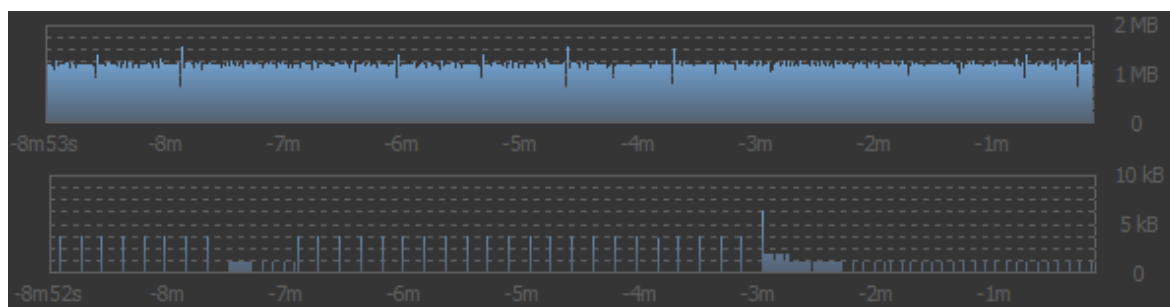
Z obrázku je patrné, že je rychlost stahování (až na výjimky) téměř konstantní, jednotlivé přenosy přímo navázané na sebe a nedochází k zbytečným prodlevám.

Stahování dat přes basicHttpBinding



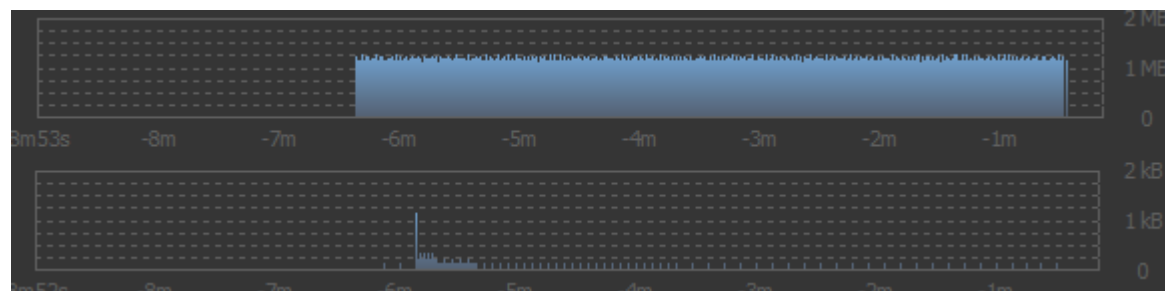
Obr. 13: Průběh stahování dat přes basicHttpBinding v čase

Mezi dotazy a odpověďmi dochází k prodlevám, přenosové pásmo není efektivně využito. Právě tento jev může být důsledkem nepříznivých časových hodnot pro basicHttpBinding. Pro přenos dat přes protokol https dochází ke stejnému jevu.

Stahování dat přes wsHttpBinding

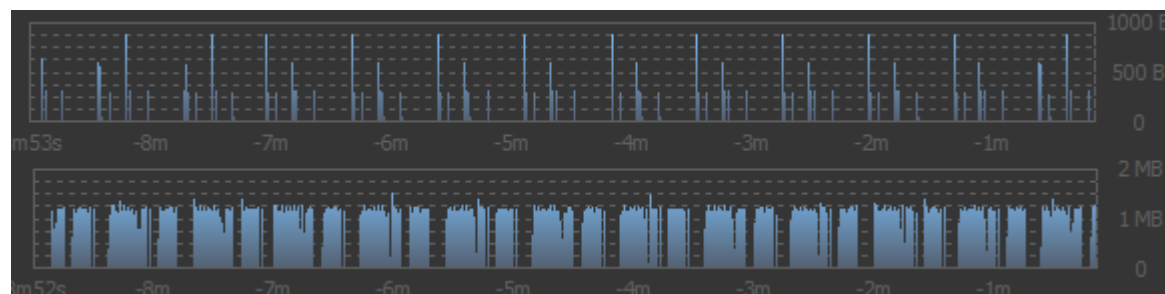
Obr. 14: Průběh stahování dat přes wsHttpBinding v čase

Mezi dotazy a odpověďmi na ně není žádná časová prodleva, při stahování se využívá celá dostupná kapacita sítě, v levé části obrázku lze pozorovat končící přenos při Message Security (větší rozestup dotazů) a vpravo začínající přenos Transport Security (menší rozestup dotazů).

Stahování dat přes netTcpBinding

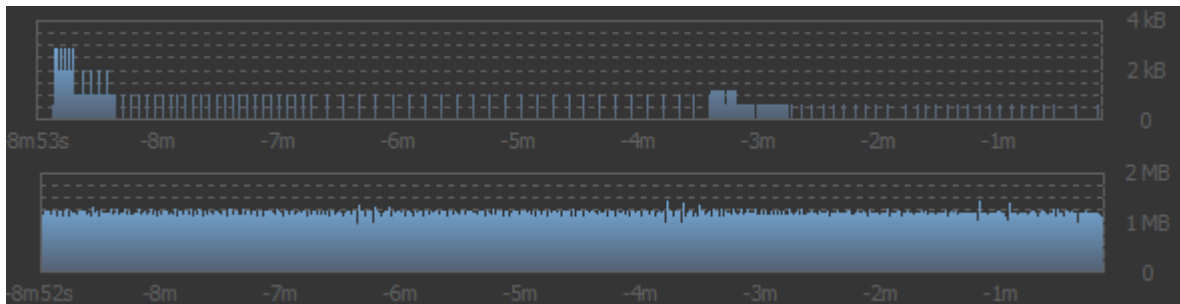
Obr. 15: Průběh stahování dat přes netTcpBinding v čase

Reakce serveru při použití netTcpBindingu okamžitě navazují na sebe. Oproti přenosu přes HTTP je průběh v čase celkově hladší. Stahování dat začíná v obrázku časově dříve než požadavek na ně, toto je způsobeno tím, že se do záznamu přimíchala dobíhající odpověď na požadavek, který byl mimo časový rozsah záznamu.

Upload dat přes WeAPI

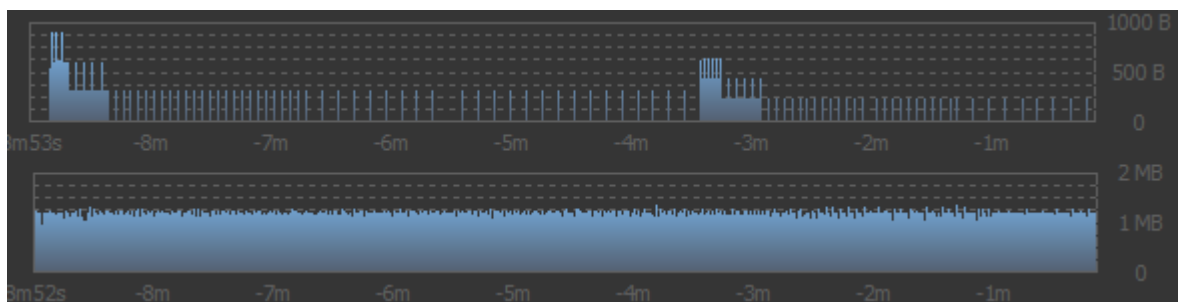
Obr. 16: Průběh nahrávání dat přes WebAPI v čase

Při uploadování dat v parametru funkce dochází v komunikaci k prodlevám.

Upload dat přes basicHttpBinding

Obr. 17: Průběh nahrávání dat přes basicHttpBinding čase

V obrázku lze pozorovat jemnou nuanci při uploadování dat mezi streamingem (vlevo) a bufferingem (vpravo). Přes oba způsoby byla přenášena stejná data (vpravo by přenos ještě dál časově pokračoval).

Upload dat přes netTcpBinding

Obr. 18: Průběh nahrávání dat přes netTcpBinding v čase

Obrázek ukazuje rozdíl mezi streamingem (vlevo) a bufferingem (vpravo). Lze si rovněž všimnout, že oproti Obr. 17 proběhl přenos stejných dat o něco rychleji.

11.2 Veřejné měření

Veřejné měření už ze své podstaty není tak přesné jako měření laboratorní. Ve veřejném měření bylo nutno zpracovat výsledky, kde každé jednotlivé měření bylo provedeno na jiné rychlosti linky, s jiným počtem opakování (dle rychlosti linky), neznámou odezvou a rychlostní stabilitou linky v čase. Měřené hodnoty vykazovaly velké časové rozdíly v jednotlivých opakováních jednoho měření. Změřené hodnoty mají tedy velmi nízkou vypovídající hodnotu.

Pro zpracování dat byla využita metoda průměrování, kdy se nejdříve vytvořil průměr z naměřených hodnot získaných opakováním dané datové jednotky při daném způsobu

přenosu dat v rámci jednoho měření, výsledkem byla průměrná hodnota pro jedno měření. Průměrné hodnoty pro jednotlivá měření se dále zprůměrovaly a vznikla výsledná hodnota.

Výsledné hodnoty pro přenos jednotlivých datových jednotek přes různé způsoby jsou uvedeny v tabulkách níže. Grafické znázornění je vytvořeno z vypočtených průměrů.

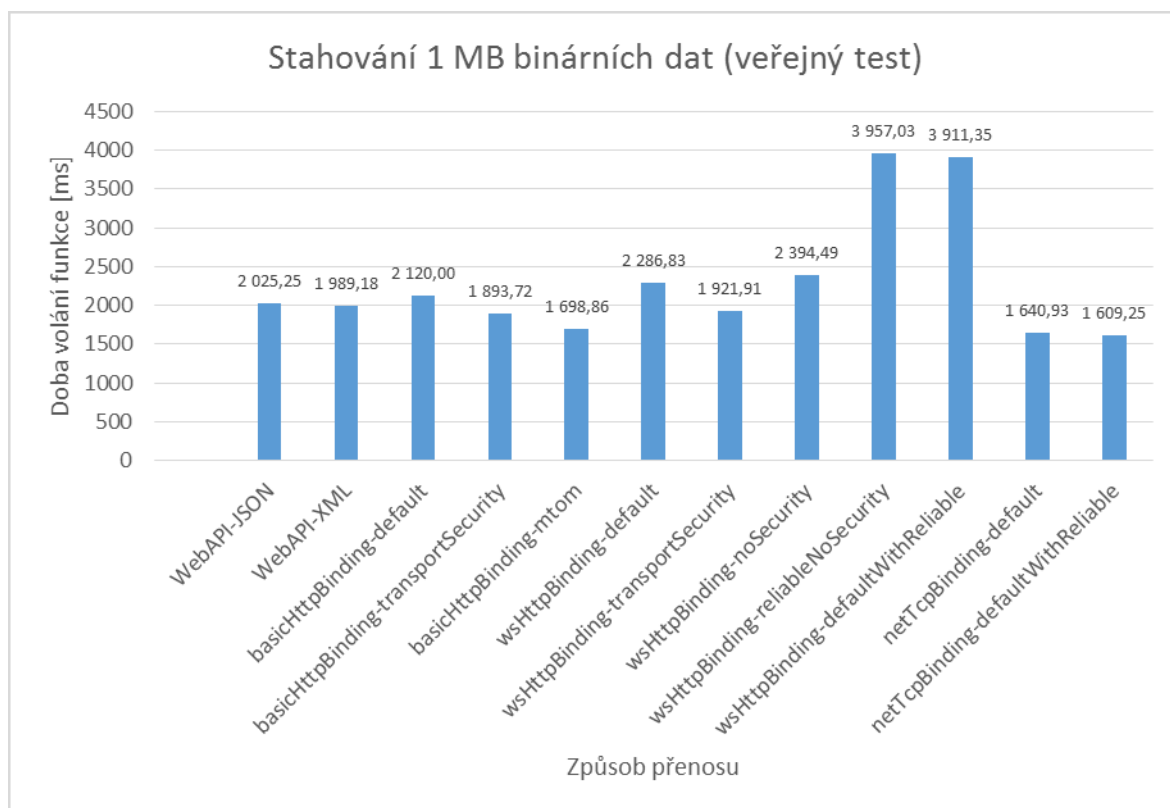
11.2.1 Výsledky testu stahování binárních dat

Při veřejném měření byly stahovány datové jednotky uvedené v *Tab. 5* způsoby uvedenými v *Tab. 6*.

Průměr [ms]	Binary 50k	Binary 100k	Binary 500k	Binary 1M
WebAPI-JSON	254,10	254,96	1045,99	2025,25
WebAPI-XML	207,73	211,53	1005,02	1989,18
basicHttpBinding- default	118,81	206,33	973,20	2120,00
basicHttpBinding- transportSecurity	162,20	271,91	962,50	1893,72
basicHttpBinding- mtom	155,85	190,51	771,82	1698,86
wsHttpBinding- default	181,98	330,53	1180,12	2286,83
wsHttpBinding- transportSecurity	200,37	302,28	1191,08	1921,91
wsHttpBinding- noSecurity	163,94	226,50	1278,36	2394,49
wsHttpBinding- reliableNoSecurity	244,49	308,09	1686,43	3957,03
wsHttpBinding- defaultWithReliable	231,08	407,51	2013,78	3911,35
netTcpBinding- default	105,29	184,15	874,72	1640,93
netTcpBinding- defaultWithReliable	84,12	170,31	772,76	1609,25

Tab. 14: Stahování binárních dat – veřejný test

Výsledky volně sledují trend laboratorního měření, jsou však velmi zkreslené, nepříznivé výsledky pro WebAPI jsou pravděpodobně způsobeny větším vlivem při ustavování TCP spojení, „záhadou“ jsou nižší časy pro Reliable Messaging u netTcpBindingu. Naměřené hodnoty reflektují spíše výkyvy při měření než rozdíly samotných protokolů.



Graf 8: Stahování 1 MB binárních dat - veřejný test (vybraná hodnota)

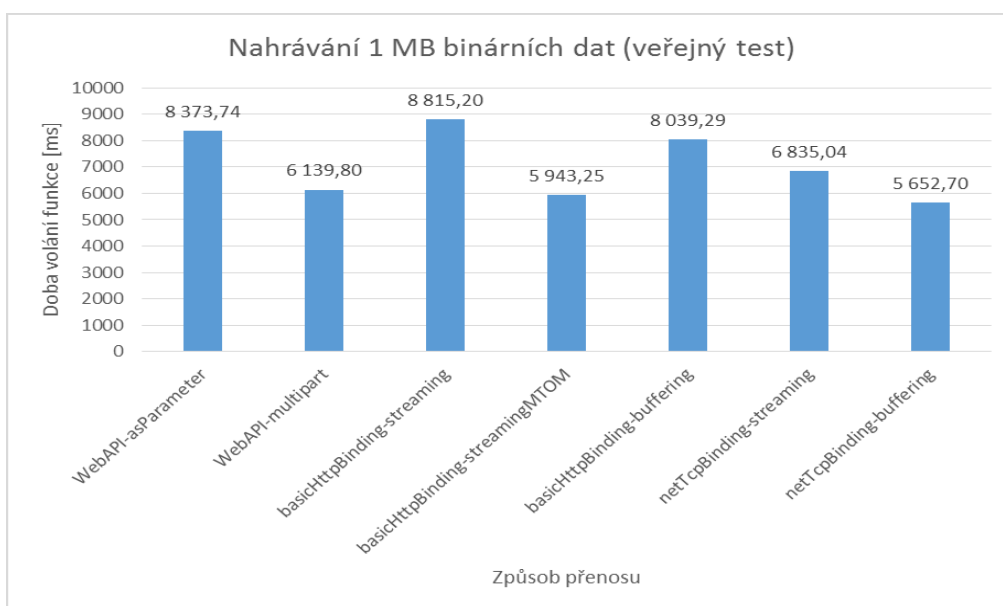
11.2.2 Výsledky testu nahrávání binárních dat

Při veřejném měření byly uploadovány datové jednotky uvedené v *Tab. 7* způsoby uvedenými v *Tab. 8*.

Průměr [ms]	Upload 50k	Upload 100k	Upload 500k	Upload 1M
WebAPI-asParameter	496,21	843,43	4202,99	8373,74
WebAPI-multipart	383,89	608,18	3120,08	6139,80
basicHttpBinding-streaming	561,20	884,86	4166,50	8815,20
basicHttpBinding-streamingMTOM	471,86	705,65	3201,69	5943,25
basicHttpBinding-buffering	409,79	760,59	3774,47	8039,29
netTcpBinding-streaming	438,54	607,59	3257,94	6835,04
netTcpBinding-buffering	306,03	649,99	2898,59	5652,70

Tab. 15: Nahrávání binárních dat – veřejný test

Trendy hodnot pro upload více kopírují výsledky laboratorního měření, pravděpodobně je to dáno tím, že upload dat probíhal na nižších rychlostech než jejich stahování, v důsledku čehož jsou časy přenosu všeobecně větší, čímž se eliminoval vliv náhodných výkyvů sítě.



Graf 9: Nahrávání 1 MB binárních dat - veřejný test (vybraná data)

12 URČENÍ REŽIE Z PŘENÁŠENÝCH DAT

Tato kapitola je věnována určení režie jednotlivých způsobů přenosu dat ze zaznamenaných dat v průběhu měření pomocí packet snifferu Wireshark.

12.1 Srovnání velikosti režie jednotlivých způsobů přenosu

První způsob určuje velikost režie jednotlivých způsobů přenosu. Vychází z binárního přenosu dat v rozmezí 1B až 1000B (testy Binary1 až Binary1k v *Tab. 1*), hodnoty jsou dále rámcově ověřeny (a případně upraveny) dle hodnot zjištěných pro jiné objemy přenášených dat. Režie je součtem režie v obou komunikačních směrech a vyjadřuje celkovou režii v rámci jednoho volání funkce.

Metodika je velmi jednoduchá a možná trochu nepřesná, protože zanedbává drobné detaily např. při kódování binárních dat do base64, kde se délka bloku dat vždy zarovná na násobek 3B apod., tato chyba je ovšem v celkovém množství dat zanedbatelná a nijak neovlivňuje pohled na způsob přenosu dat jako na celek. Metodika poskytuje spolehlivý postup, jak hodnotit režii daného způsobu přenosu dat jako celek.

Podstatou je výpočet celkového množství přenášených dat na aplikační vrstvě v rámci souhrnného testu, zjištění množství dat, která byla přenášena v rámci zachyceného TCP streamu daného testu, odečtení těchto dvou hodnot od sebe a vydělení výsledné hodnoty celkovým počtem dílčích přenosů (volání funkce) v rámci daného testu, čímž se získá režie pro jeden přenos, který proběhl daným způsobem. Vždy je nutno vycházet z povahy přenášených dat a při výpočtu zohlednit tzv. statickou režii, která je přidána ke každému přenosu vždy (volání funkce) a tzv. dynamickou režii, která je závislá na použitém kódování a množství přenášených dat. Dynamická režie se uplatňuje pouze u přenosu binárních dat kódovaných pomocí base64.

Princip lze ilustrovat vztahem níže, v praxi je ale nutno výpočet přizpůsobit povaze samotného testu a datům zachyceným v TCP streamu.

Pokud jsou data kódována base64:

$$sr = \frac{dts - dav \cdot 1,33}{ppt},$$

kde sr – statická režie jednoho přenosu, dts – velikost dat v TCP streamu, dav – suma dat na aplikační vrstvě v kontextu zachycené komunikace v daném TCP streamu, ppt – počet provedených přenosů v kontextu zachycené komunikace v daném TCP streamu, dynamická režie bude vždy přibližně 33%

Pro data, která jsou přenášena binárně:

$$sr = \frac{dts - dav}{ppt},$$

význam zástupných symbolů je stejný jako v předchozím případě, dynamická režie je 0%.

Princip výpočtu je vhodné ilustrovat na praktickém příkladu:

Výpočet režie přenosu dat přes WebAPI:

Fakta:

- Přenášené (měřené) velikosti dat 1 B, 5 B, 10 B, 50 B, 100 B, 500 B, 1000 B (tj. 1666 B dat na aplikační vrstvě, 7 přenosů).
- Data byla kódována base64 (tj. velikost dat při přenosu je přibližně 1,33 násobek původní velikosti).
- Test proběhl pro JSON i XML 25x (tj. 25 x 2 formáty).
- Před samotným měřením došlo při každém opakování k „protažení TCP“ spojení naprázdno velikostí dat 100B, 100B, 100B, 100B, 1000B (tj. 1400 B dat, 5 přenosů v každém opakování).
- Zachycený TCP stream má délku 364 474 B a obsahuje vše, co je uvedené v bodech výše.

Výpočet:

$$sr = \frac{364474 - (1666 \cdot 2 \cdot 25 + 1400 \cdot 25) \cdot 1,33}{(7 \cdot 2 + 5) \cdot 25} \approx 436 B$$

Statická režie pro jeden přenos WebAPI je tedy přibližně 436 B, dynamická režie je 33%. Vypočtené hodnoty byly dále korigovány měřením jiného objemu dat, zde vyšla statická režie pro WebAPI 515 B, průměrná režie je tedy 475 B. Nepřesnosti mohou být způsobeny

zkreslenou informací o velikosti zachyceného streamu. Hodnoty režie pro vybrané způsoby přenosu jsou uvedeny v tabulce níže.

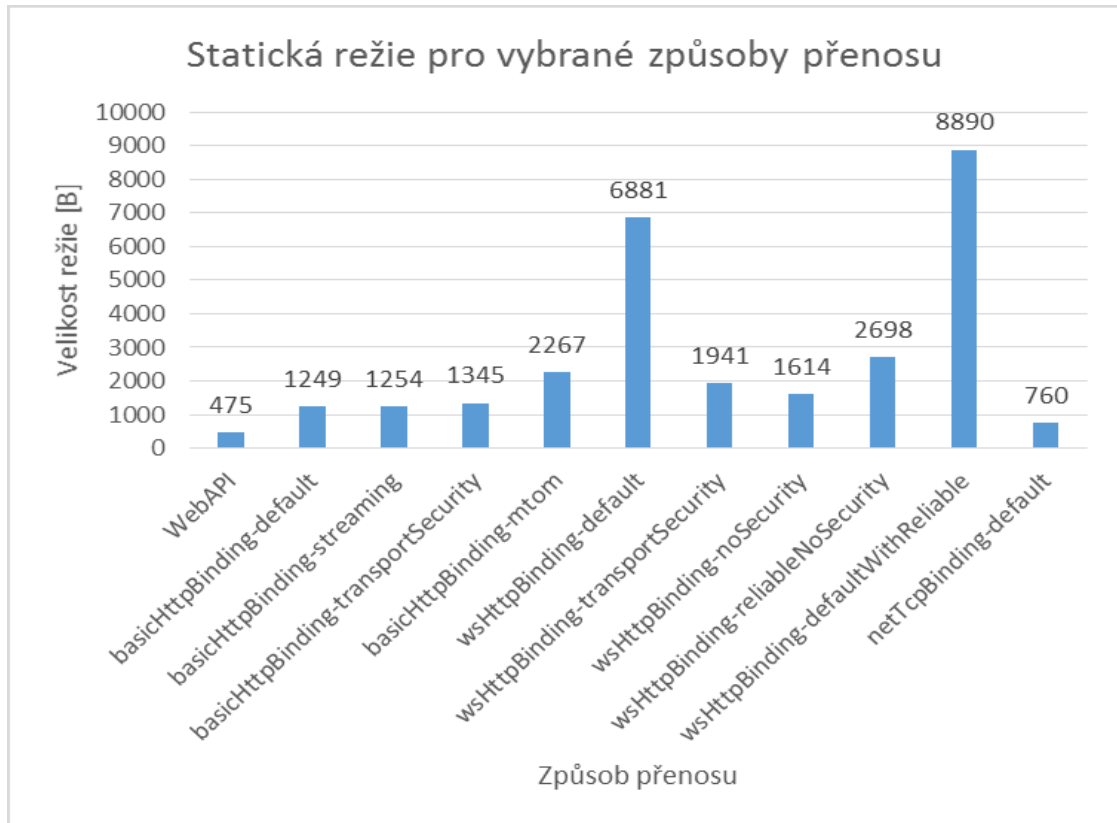
	Statická režie [B]	Dynamická režie [%] ⁷
WebAPI	475	33
basicHttpBinding-default	1249	33
basicHttpBinding-streaming	1254	33
basicHttpBinding-transportSecurity	1345	33
basicHttpBinding-mtom	2267	0
wsHttpBinding-default	6681	33
wsHttpBinding-transportSecurity	1941	33
wsHttpBinding-noSecurity	1614	33
wsHttpBinding-reliableNoSecurity	2698	34
wsHttpBinding-defaultWithReliable	8890	34
netTcpBinding-default	760	0
udpBinding-default	13242	1000

Obr. 19: Průměrná režie při volání funkce dle způsobu přenosu

Z vypočtených hodnot vyplývá, že nejnižší průměrnou režii na jedno volání funkce má WebAPI a to 475 B, při přenosu binárních dat ovšem přidává režii 33% velikosti přenášených dat (hodnota pro WebAPI byla zjištěna z dat zachycených při testování stahování binárních dat). Z WCF bindingů má nejnižší režii netTcpBinding, který navíc ve výchozím nastavení využívá TLS, a to 760 B, při posílání binárních dat zde dále nedochází k žádné další režii. Průměrná režie komunikace přes TLS (http vs. https) je pro jednu zprávu přibližně 211 B (odhadem 10% pro krátkou životnost TLS, 1,38 % vůči celkové délce komunikace). MTOM u basicHttpBindingu přidává speciální hlavičku do každé zprávy, ale velikost samotných dat nepřináší další režii. Message Security přidává režii přibližně 5067 B, Reliable Messaging přibližně 1084-2009 B (přibližně 30 % pro objemy dat okolo 300 B, 4% při objemech dat okolo 300 kB). Použití streamingu oproti bufferingmu přináší průměrnou režii okolo 1%.

K vypočteným hodnotám je nutno přistupovat pouze orientačně, jedná se o průměrné hodnoty, samotná režie se může u jednotlivých zpráv mírně lišit.

⁷ Dynamická režie se uplatňuje pouze u přenosu binárních dat.



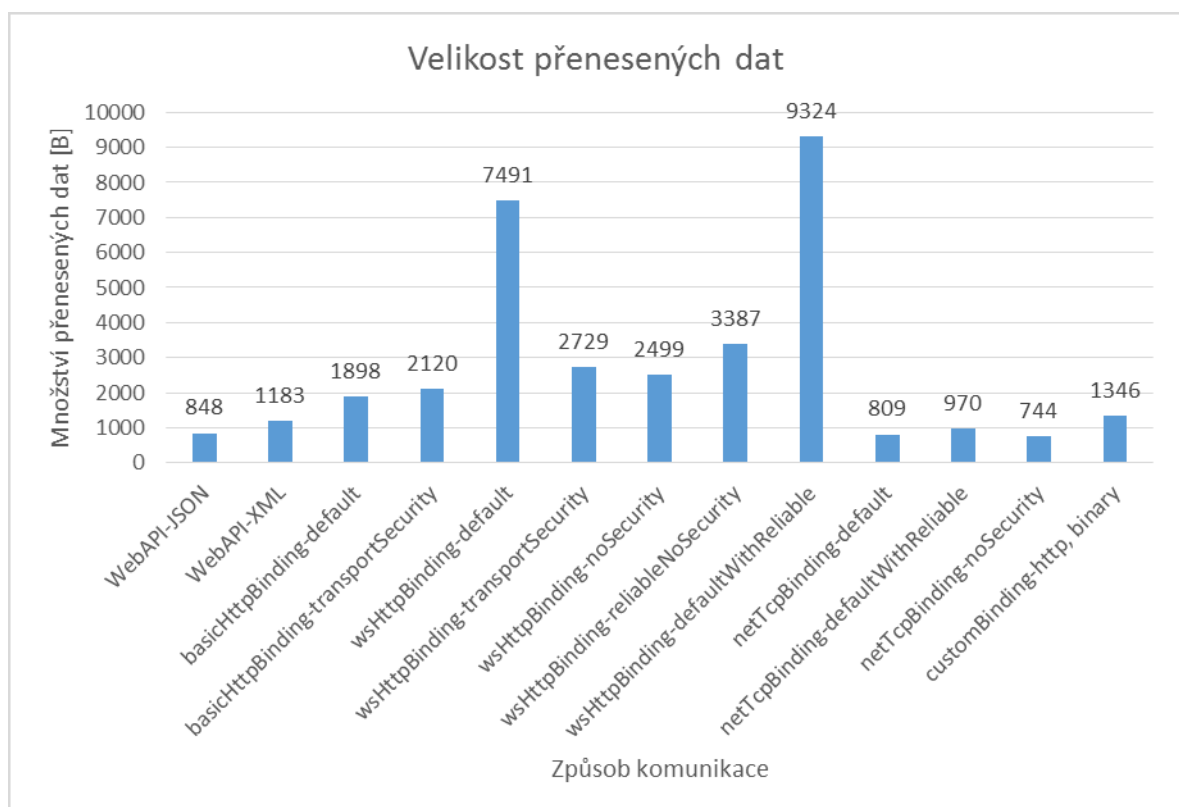
Graf 10: Průměrná režie při volání funkce dle způsobu přenosu

12.2 Srovnání velikosti přenesených dat při volání funkce

Tabulka níže obsahuje celkové množství dat, které bylo přeneseno při volání funkce, která jako odpověď vrací objekt s 40 proměnnými. Funkce je volána přes různé způsoby komunikace. Uvedené hodnoty jsou součtem velikosti dat pro dotaz i odpověď (v odpovědi se nachází objekt se 40 proměnnými). Cílem je poukázat na režii rozdílného způsobu serializace objektu a obsah zprávy. Samotná komunikace, ze které jsou hodnoty odvozeny, je uvedena v příloze P III na konci této práce.

Komunikace	Velikost přenesených dat [B]
WebAPI-JSON	848
WebAPI-XML	1183
basicHttpBinding-default	1898
basicHttpBinding-transportSecurity	2120
wsHttpBinding-default	7491
wsHttpBinding-transportSecurity	2729
wsHttpBinding-noSecurity	2499
wsHttpBinding-reliableNoSecurity	3387
wsHttpBinding-defaultWithReliable	9324
netTcpBinding-default	809
netTcpBinding-defaultWithReliable	970
netTcpBinding-noSecurity	744
customBinding-http, binární serializace	1346

Obr. 20: Množství přenesených dat při volání funkce



Graf 11: Množství přenesených dat při volání funkce

12.2.1 Rozdíl mezi vyhodnocením v kap. 12.1 a 12.2.

V kapitole 12.1 je analyticky vypočítána průměrná čistá režie, kterou přidá samotný protokol. Vychází se z přenosu různých kombinací binárních dat. Od množství přenesených dat na transportní vrstvě je odečteno množství dat na aplikační vrstvě a rozdíl je podělen celkovým počtem volání různých funkcí, které jsou zachyceny v daných datech.

V kapitole 12.2 jsou uvedeny velikosti, které byly zjištěny z obsahu komunikace týkající se jednoho volání funkce, která vrací objekt se 40 proměnnými. Vzal se tedy samotný obsah komunikace daného jednoho volání (uveden v příloze P III) a zjistila se jeho délka.

Objekt se 40 proměnnými byl volen z důvodu zohlednění režie serializace jako zástupce běžně přenášených dat.

Vyhodnocení v kap. 12.1 a 12.2 jsou tedy provedena nezávisle na sobě a diametrálně odlišným způsobem. Každé rovněž udává hodnoty pro jiný účel. Běžným pohledem lze zpozorovat podobný trend v obou vyhodnoceních.

13 DOPORUČENÍ ZPŮSOBU PŘENOSU DAT

Doporučení nejvhodnějšího způsobu, jak přenášet data je nutno rozdělit na dvě části – doporučení pro komunikaci kdy obě strany běží na technologii .NET a doporučení pro kompatibilní komunikaci technologie .NET s jinou technologií, například s mobilním klientem na platformě Android nebo iOS.

13.1 Komunikace mezi zařízeními podporujícími .NET

Pro komunikaci, kde je na obou stranách technologie .NET lze s výhodou využít binárního `netTcpBinding` technologie WCF. Tento způsob přenosu vyhrává „na plné čáře“ před všemi ostatními způsoby z hlediska celkového času potřebného pro volání funkce. Datová režie je vzhledem k výchozí podpoře komunikace přes TLS tunel téměř srovnatelná s WebAPI, navíc zde nedochází ke zvětšování objemu přenášených binárních dat. `NetTcpBinding` lze se zapnutým TLS (Transport Layer Security) s výhodou provozovat na protokolu 443 (pokud to způsob hostování umožňuje) a tím přesvědčit ostatní mezilehlá zařízení, že probíhá běžná HTTPS komunikace. Rozdíl oproti HTTPS zde není podstatný, protože po vytvoření TCP spojení je toto transparentně zašifrováno protokolem TLS a mezilehlá zařízení nemají prostředky jak poznat, zda přes šifrovaný tunel probíhá HTTP nebo jiná komunikace a nebudou ji bránit. `NetTcpBinding` využívá certifikát nastavený na úrovni služby, kdežto HTTPS využívá certifikát nastavený na úrovni webového hostingu, na samotný TLS tunel to ale nemá vliv. Při tomto způsobu přenosu lze rovněž autentizovat klienta (mimo jiné) pomocí certifikátu, čímž lze vzájemně ověřit identitu obou komunikujících stran a bezpečně šifrovat přenášená data. V případě, že provoz (při použití TLS) není uskutečňován na portu 443, může být blokován mezilehlými zařízeními (firewall), zde je nutno spolupracovat se správcem těchto zařízení a zajistit průchodnost dat.

13.2 Komunikace se zařízením nepodporující .NET

Pro komunikaci se zařízením, které nepodporuje .NET je nutno stanovit, jaké jsou požadavky na komunikaci.

V případě, že je jediným požadavkem minimální režie, jako nejvhodnější způsob přenosu (mimo nahrávání souboru) se jeví použití WebAPI se serializací do formátu JSON. Tuto komunikaci lze volitelně zabezpečit pomocí TLS a data přenášet přes HTTPS.

V případě existující infrastruktury na technologii WCF a požadavků na nejnižší režii je nejvhodnější použít `basicHttpBinding`, volitelně přes protokol HTTPS (Transport Security) a rovněž volitelně (dle převažující povahy dat) s optimalizací pro přenos binárních dat (MTOM). Naměřená data ovšem ukazují, že provoz přes `basicHttpBinding` není efektivní z hlediska využití sítě.

Pro efektivní využití sítě a stabilní odezvu služby, případně pro využití Reliable Messagingu je vhodné použít `wsHttpBinding` se zabezpečením Transport Security (HTTPS), pokud není šifrování dat důležité, je možno zabezpečení vypnout a používat protokol HTTP, čímž se ušetří část režie.

V případě, že je potřeba přenášet zabezpečené zprávy v rozlehlém distribuovaném systému, kde dochází ke směrování zpráv mezi jednotlivými uzly, případně jsou zprávy přenášeny v různých částech systému na rozdílných protokolech na nižší vrstvě, je nutno zprávu přenášet pomocí `wsHttpBindingu` a zabezpečit samotný obsah zprávy pomocí Message Security.

Při nahrávání binárních dat na server (uploadu souboru) je vhodné využít technologii WebAPI a uploadovat soubory jako Multipart/form-data, přenos lze zabezpečit přes HTTPS. Pokud již existuje infrastruktura postavená na technologii WCF, je možné nahrávat binární data pomocí `basicHttpBindingu` nebo `wsHttpBindingu` s optimalizací MTOM a volitelně se zpracováním dat jako stream. Pokud je požadavek zpracovávat data jako stream, musí být k tomu uzpůsobeny metody tak, aby pracovaly s daty jako se streamem, dále je nutno využít MessageContract, kde se v těle zprávy specifikuje daný stream.

14 PRAKTICKÝ PŘÍNOS PRÁCE

Výsledky práce byly využity ve společnosti Business Logic s.r.o. Tato společnost se zabývá komplexními řešeními na míru v oblasti webových stránek, desktopových a mobilních aplikací. Častým požadavkem zákazníka je rozsáhlý systém, který je zpřístupněn na rozdílných platformách. Pro tyto účely je zvláště vytvořena serverová část poskytující služby a dále pak klientské části na jednotlivých systémech, které s těmito službami pracují.

Se vzrůstajícím počtem připojených klientů k serveru začalo docházet k problému, kdy se celková doba komunikace začala značně protahovat, což snižovalo pracovní efektivitu uživatelů využívající systém. Snahou tedy bylo optimalizovat komunikaci serveru s klienty tak, aby se maximálně snížil objem přenášených dat.

Společnost hledala nejvhodnější způsob přenosu dat při využití již implementované technologie WCF (SOA), padaly rovněž návrhy na přepsání systému do technologie WebAPI (REST). To, co chybělo, byla reálná data, na základě kterých by mohlo padnout finální rozhodnutí. Všude na Internetu byla pouze všeobecná doporučení, nikoliv reálná naměřená data.

Z tohoto důvodu byla vypracována tato práce, která poskytuje komplexní srovnání technologií z hlediska času potřebného na volání funkce a režie přenosu pro různé typy dat a různé komunikační způsoby.

Na základě dat zjištěných zde byla komunikace desktopové části aplikace převedena z technologie WCF WsHttpBindingu se zabezpečením na transportní vrstvě na WCF Net.Tcp Binding, čímž se znatelně snížily odezvy klientské části systému. Klient nyní při pokusu připojení k serveru vyzkouší Net.Tcp Binding a pokud a je pokus úspěšný (komunikace není po cestě blokována), použije pro připojení k serveru tento způsob. Pokud se připojení přes Net.Tcp nepodaří, klient využije původního WsHttpBindingu.

Logy ukazují, že nového způsobu připojení využívá 98% klientů, čímž se podařilo snížit objem původní datové komunikace přibližně na 30%. Uvolnila se tak síťová kapacita pro podporu dalších klientů vstupujících do systému a to při zachování původní síťové infrastruktury.

Podpora klientů na jiných platformách byla dočasně vyřešena pomocí `basicHttpBinding` se zabezpečením na transportní vrstvě, spekuluje se o využití technologie REST se serializací objektu do formátu JSON a to jak pomocí `WebHttpBinding` technologie WCF, jež umožňuje využít stávající infrastrukturu služeb a poskytovat ji přes principy REST na protokolu HTTP, tak i přes „čistokrevnější“ REST framework WebAPI. Rozdíl mezi těmito technologiemi není ve formátu přenášených dat, ale v rozdílné implementaci kódu na straně serveru. V době psaní této práce stále probíhala evaluace a testování těchto technologií.

Práce poskytuje komplexní srovnání velkého množství různých způsobů přenosu dat pro různé typy dat za konstantních síťových podmínek, které může být využito kýmkoliv, komu nestačí jednoduchá tvrzení kolující Internetem a částečné neúplné testy omezeného množství způsobů komunikace provedené za nejasných podmínek objevující se na blozích uživatelů.

V příloze práce P III jsou uvedeny ukázky přenášených dat, což pomáhá ilustrovat komplexnost a formát přenášených dat u jednotlivých způsobů komunikace.

ZÁVĚR

Tato práce se věnovala srovnání výkonu různých typů komunikace u distribuovaných aplikací. V teoretické části bylo podrobně rozebráno, co jsou to distribuované aplikace, jaké datové formáty se pro komunikaci používají, přes jaké protokoly na nižších vrstvách komunikace probíhá, různé způsoby komunikace z hlediska architektury, popis technologií zajišťujících komunikaci pro distribuované aplikace v prostředí .NET. Dále byly rozebrány vyhodnocovací nástroje pro analýzu síťové komunikace.

V praktické části práce bylo navrženo testovací prostředí a vytvořen testovací program v prostředí .NET. Následně byly provedeny testy v laboratorním prostředí, kde byla přísně kontrolována přenosová rychlost a opakovaně měřeny časy přenosu různých jednotek dat přes různé způsoby komunikace. Testován byl přenos různě složitěho objektu, přenos stahování různé velikosti binárních dat a nahrávání souboru na server. Testovány byly frameworky WebAPI, serializace do JSON a XML, a WCF, různé formy nastavení basicHttp, wsHttp a netTcp bindingu testující vlivy samotných bindingů, různých způsobů zabezpečení komunikace, využití garantovaného doručování zpráv v pořadí odeslání, různých způsobů přenosu binárních dat a různých způsobů zpracování zpráv na straně serveru. Z důvodů ověření výsledků laboratorních testů byla vytvořena sada veřejných testů, které byly po dobu jednoho týdne dostupné pro veřejnost a každý si mohl stáhnout testovacího klienta a provést test přes svoji Internetovou linku oproti veřejnému testovacímu serveru. Testu se účastnilo 9 lidí. Naměřená data laboratorního i veřejného měření byla statisticky vyhodnocena a přehledně prezentována ve formě tabulek a grafů, které byly následovány textovým rozбором nejdůležitějších poznatků vyplývajících z výsledků daného testu. Data v průběhu jednotlivých laboratorních testů byla zaznamenána packet snifferem a následně z nich byla matematicky odvozena průměrná režie jednotlivých způsobů přenosu v bytech. Povaha síťové komunikace jednotlivých způsobů přenosu byla rovněž zachycena, toto umožnilo vysvětlit odchylky, které se objevily v naměřených datech a dále zhodnotit výkonnost způsobu komunikace z hlediska efektivity využití sítě. Na základě teoretických poznatků, praktických zkušenosti z průběhu testování, výsledků testů, vypočítaných hodnot režie a znalosti chování přenosu z hlediska efektivnosti využití dostupné kapacity sítě byla sestavena sada doporučení, která pomohou na základě klíčových požadavků vybrat nejvhodnější způsob komunikace a tím značně uspořit kapacitu přenosového média a snížit odezvu inter-aplikační komunikace.

Práce našla praktické uplatnění již v průběhu psaní, kdy na jejím základě došlo k úpravě softwaru vyvíjeného ve společnosti Business Logic s.r.o., čímž se podařilo snížit celkový objem přenášených dat a tím i zvýšit rychlost odezvy klientských aplikací.

ZÁVĚR V ANGLIČTINĚ

This thesis was dedicated to comparison of performance of different types of communication in distributed applications. In theoretical part, there was elaborately examined what are distributed applications, which data formats are used for communication, which lower layer protocols communication uses, different communication types from the point of view of architecture and description of technologies facilitating communication for distributed applications in .NET environment. There is also description of network analyzing tools.

In practical part, there was designed testing environment and created test application in NET environment. Subsequently, there were performed tests in laboratory conditions, in which network speed was strictly controlled. Tests have been carried out repeatedly and there was measurement of time which has been taken by transfer for each of many different data units transferred by many different ways of communication. Tested was transfer of object with various complexity, transfer of various sizes of binary data and file upload to server. Tested were frameworks WebAPI, serialization to JSON and XML, and WCF, different ways of setting of basicHttp, wsHttp and netTcp bindings, testing influence of bindings itself, different ways of secure communication, use of reliable messaging to deliver messages in the same order they were sent, different ways of binary data transfer and different ways of message processing on server side. Owing to verify results of laboratory tests, there was created set of public tests, which was for period of one week accessible to public and everybody could download test client and perform test by using his/her Internet connection to public test server. Test was ran by 9 people. Recorded values of laboratory and public tests were statistically processed and clearly presented in the form of tables and graphs, which were followed by text breakdown of the most important finding from results of given test. Data going via network were captured by packet sniffer and subsequently used to mathematically derive average overhead in bytes of respective ways of communication. Behavior of network data flow in time was also captured, this helped to clarify deviations which were found in recorded values and evaluate performance of different ways of communications from the point of view of network utilization affectivity. Based on theoretical information, practical experiences from the testing, test results, computed values of overhead and knowledge of behavior of data transfer in time, was put together set of recommendations, which help to choose right way

of communication based on key requirements and by this substantially save network capacity and reduce response in inter-application communication.

The work has been practically used already in time of writing. Based on this thesis, there has been made changes in software developed in company Business Logic s.r.o. This changes helped reduce overall volume of transferred data and thus increase responsiveness of client part of application.

SEZNAM POUŽITÉ LITERATURY

- [1] *MSDN magazine*. Chatsworth, CA: 1105 Media, Inc., roč. 2012, NUMBER 10. ISSN 1528-4859. Dostupné z: http://download.microsoft.com/download/6/3/E/63EEE0EA-D7A8-4BE6-B71C-5E3DFDCE18F9/MDN_1012DG.pdf
- [2] PLURALSIGHT LLC. *The Elements of Distributed Architecture: Failure Management* [online]. 2011 [cit. 2013-02-25]. Dostupné z: <http://pluralsight.com/training/courses/TableOfContents?courseName=eda>
- [3] PLURALSIGHT LLC. *WCF Fundamentals* [online]. 2008 [cit. 2013-03-23]. Dostupné z: <http://pluralsight.com/training/courses/TableOfContents?courseName=wcf-fundamentals>
- [4] TROELSEN, Andrew. *Pro C# and the .NET 4.5 framework*. 6. edition. Berkeley, California: APress, 2012. ISBN 978-143-0242-338.
- [5] SOLIS, Daniel. *Illustrated C# 2012*. 4th edition. New York: Apress, 2012, 764 s. ISBN 978-1-4302-4279-6.
- [6] PLURALSIGHT LLC. *XML Fundamentals* [online]. 2012 [cit. 2013-03-09]. Dostupné z: <http://pluralsight.com/training/courses/TableOfContents?courseName=xml-fund>
- [7] Extensible Markup Language (XML) 1.0: REC-xml-19980210. *W3C* [online]. 1998 [cit. 2013-03-09]. Dostupné z: <http://www.w3.org/TR/1998/REC-xml-19980210>
- [8] Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C* [online]. 2008, 7.2.2013 [cit. 2013-03-09]. Dostupné z: <http://www.w3.org/TR/2008/REC-xml-20081126/>
- [9] XML Namespaces. *W3Schools* [online]. 2013 [cit. 2013-03-09]. Dostupné z: http://www.w3schools.com/xml/xml_namespaces.asp
- [10] XSD - The <schema> Element. *W3Schools* [online]. 2013 [cit. 2013-03-09]. Dostupné z: http://www.w3schools.com/schema/schema_schema.aspx
- [11] Introducing JSON. *JSON* [online]. 2013 [cit. 2013-03-10]. Dostupné z: <http://www.json.org>
- [12] JSON Syntax. *W3Schools* [online]. 2013 [cit. 2013-03-10]. Dostupné z: http://www.w3schools.com/json/json_syntax.asp

- [13] KOZIEROK, Charles M. *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. San Francisco: No Starch Press, c2005, 1616 p. ISBN 15-932-7047-X.
- [14] *Gigabit (1000 Mbps) Ethernet: IEEE 802.3z (1000Base-X), 802.3ab (1000Base-T) and GBIC*[online]. 2013 [cit. 2013-03-16]. Dostupné z: <http://www.javvin.com/protocolGigE.html>
- [15] PLURALSIGHT LLC. *REST Fundamentals* [online]. 2012 [cit. 2013-03-19]. Dostupné z: <http://pluralsight.com/training/courses/TableOfContents?courseName=rest-fundamentals>
- [16] 9 Method Definitions. *W3C* [online]. 2013 [cit. 2013-03-19]. Dostupné z: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9>
- [17] 10 Status Code Definitions. *W3C* [online]. 2013 [cit. 2013-03-19]. Dostupné z: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>
- [18] TLS overhead. *Netsekure rng* [online]. 2010 [cit. 2013-03-19]. Dostupné z: <http://netsekure.org/2010/03/tls-overhead/>
- [19] SOAP Version 1.2 Part 0: Primer (Second Edition). *W3C* [online]. 2007 [cit. 2013-03-21]. Dostupné z: <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>
- [20] PLURALSIGHT LLC. *WCF For Architects* [online]. 2012 [cit. 2013-03-19]. Dostupné z: <http://pluralsight.com/training/courses/TableOfContents?courseName=wcf-for-architects>
- [21] SOAP MTOM. *IBM* [online]. 2007 [cit. 2013-03-21]. Dostupné z: <http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/index.jsp?topic=%2Fcom.ibm.webservice.wsfp.doc%2Ftopics%2Fcmtom.html>
- [22] Web Services Description Language (WSDL) 1.1. *W3C* [online]. 2001 [cit. 2013-03-21]. Dostupné z: <http://www.w3.org/TR/wsdl>
- [23] GALLOWAY, Jon Alan. *Professional asp.net MVC 4*. 1st ed. Indianapolis, IN: Wiley Pub. Inc., 2012, p. cm. ISBN 11-183-4846-X.
- [24] Routing in ASP.NET Web API. *ASP.NET* [online]. 2012 [cit. 2013-03-22]. Dostupné z: <http://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api>

- [25] HTTP Message Handlers. *ASP.NET* [online]. 2012 [cit. 2013-03-22]. Dostupné z: <http://www.asp.net/web-api/overview/working-with-http/http-message-handlers>
- [26] Media Formatters. *ASP.NET* [online]. 2012 [cit. 2013-03-22]. Dostupné z: <http://www.asp.net/web-api/overview/formats-and-model-binding/media-formatters>
- [27] SHARP, John. *Windows Communication Foundation 4 step by step*. Sebastopol, Calif.: Microsoft/O'Reilly, c2010, xxix, 700 p. ISBN 07-356-4556-6.
- [28] Message Queuing (MSMQ). *MSDN* [online]. 2013 [cit. 2013-03-23]. Dostupné z: [http://msdn.microsoft.com/en-us/library/windows/desktop/ms711472\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms711472(v=vs.85).aspx)
- [29] *MSDN magazine* [online]. 2012, vol. 27, no. 12 [cit. 2013-01-27]. ISSN 1528-4859. Dostupné z: http://download.microsoft.com/download/6/3/E/63EEE0EA-D7A8-4BE6-B71C-5E3DFDCE18F9/MDN_1212DG.pdf
- [30] The WebSocket Protocol. *IETF* [online]. 2011 [cit. 2013-03-23]. Dostupné z: <http://tools.ietf.org/html/rfc6455>
- [31] *Wireshark User's Guide: for Wireshark 1.9* [online]. 2012 [cit. 2013-01-27]. Dostupné z: <http://www.wireshark.org/download/docs/user-guide-a4.pdf>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

B	.Byte. Jednotka určující množství dat. Byte se skládá z 8 bitů, označených jako b. Jeden bit obsahuje binární informaci 0 nebo 1.
DTD	Data Type Definition, definice typu dat. Prostředek umožňující formálně popsat strukturu XML dokumentu.
HTTP	HyperText Transfer Protocol, protokol pro přenos HyperTextu. Bezstavový textový protokol pro komunikaci mezi klientem a serverem využívající principu dotaz-odpověď.
IP	Internet Protocol, protokol Internetu. Zajišťuje adresování a směrování dat mezi různými sítěmi.
JSON	JavaScript Object Notation, zápis objektu JavaScriptu. Způsob, jakým je objekt uložen do textové podoby. Struktura uloženého objektu je velmi jednoduchá a datově nenáročná.
kbit/s	Kilobits per second, kilobitů za sekundu. Udává počet přenesených bitů v násobcích 1000 (nebo 1024) mezi jednotlivými částmi systému za jednu sekundu.
Mbit/s	Megabits per second, megabitů za sekundu. Udává počet přenesených bitů v násobcích 1 000 000 (nebo 1 048 576) mezi jednotlivými částmi systému za jednu sekundu.
MIME	Multipurpose Internet Mail Extensions, víceúčelová rozšíření Internetové pošty. Textový identifikátor, určující, jak přenášená binární data po přijetí interpretovat.
MSMQ	Microsoft Message Queuing, Microsoft zařazování zpráv. Implementace principu fronty zpráv firmou Microsoft. Fronta zpráv umožňuje doručovat zprávy i v případě, že je příjemce v čase odeslání zprávy nedostupný. Jedná se o tzv. asynchronní způsob komunikace.
MTOM	Message Transmission Optimization Mechanism, mechanismus optimalizace přenosu zprávy. Binární data jsou ve zprávě uloženy v binární formě a identifikovány MIME.

REST	Representational State Transfer, reprezentační přenos stavu. Aplikace postavené na tomto druhu komunikace používají zdrojově orientovanou architekturu ROA.
ROA	Resource Oriented Architecture, zdrojově orientovaná architektura. Aplikace je modelována jako sada stavů, mezi kterými klient přechází. Každý stav je reprezentován zdrojem.
RPC	Remote Procedure Call, vzdálené volání procedur. Funkcionalita, která umožňuje provádět procedury na vzdáleném výpočetním uzlu.
SOA	Service Oriented Architecture, servisně orientovaná architektura. Aplikace je modelována jako sada metod, které lze vzdáleně volat.
SOAP	Simple Object Access Protocol, jednoduchý protokol pro přístup k objektům. Protokol umožňuje přenášet objekty mezi dvěma stranami a volat funkce na vzdáleném výpočetním uzlu. Komunikace je realizována pomocí zasílání zpráv.
TCP	Transmission Control Protocol, protokol řídící přenos. Protokol slouží jako mezivrstva mezi aplikací a protokolem IP: Zodpovídá za garantované doručení dat v pořadí, v jakém byla odeslána.
TLS	Transport Layer Security, bezpečnost transportní vrstvy. Protokol je umístěn mezi aplikací a TCP, transparentně šifruje přenášená aplikační data a předává je TCP. Na základě asymetrické kryptografie je vytvořen symetrický klíč, který se používá na šifrování přenášených dat.
UDP	User Datagram Protocol, uživatelský datagramový protokol. Protokol slouží jako mezivrstva mezi aplikací a protokolem IP. Protokol nezajišťuje doručení dat, data se mohou po cestě ztratit nebo dojít v jiném pořadí než byla odeslána. Používá se k přenosu nedůležitých dat, která jsou časově citlivá.
URI	Uniform Resource Identifier, unikátní identifikátor zdroje. Identifikuje zdroj v rámci architektury ROA.

- WCF Windows Communication Foundation, základ komunikace Windows. Unifikovaný framework pro systémy Windows umožňující vzdáleně volat procedury na jiném zařízení. Umožňuje zpřístupnit jednu aplikační implementaci přes různé protokoly.
- WSDL Web Services Description Language, jazyk pro popis webových služeb. Umožňuje popsat, jaké možnosti server v rámci dané služby poskytuje.
- XML Extensible Markup Language, rozšiřitelný značkovací jazyk. Textový způsob popisu objektů. Jazyk je základem pro formát zprávy SOAP.

SEZNAM OBRÁZKŮ

<i>Obr. 1: Organizace distribuovaného systému pro sdílení výkonu [1, s. 20]</i>	14
<i>Obr. 2: .NET – průběh kompilace [5]</i>	19
<i>Obr. 3: TCP Segment [13]</i>	32
<i>Obr. 4: IP Packet [13]</i>	34
<i>Obr. 5: Ethernet rámeček</i>	35
<i>Obr. 6: Okno programu Wireshark</i>	58
<i>Obr. 7: Okno programu Cascade Pilot</i>	59
<i>Obr. 8: Okno programu NetLimiter</i>	60
<i>Obr. 9: Okno self-hostované WCF služby</i>	74
<i>Obr. 10: Testovací klient - laboratorní měření</i>	77
<i>Obr. 11: Testovací klient – veřejné měření</i>	78
<i>Obr. 12: Průběh stahování dat přes WebAPI v čase</i>	90
<i>Obr. 13: Průběh stahování dat přes basicHttpBinding v čase</i>	90
<i>Obr. 14: Průběh stahování dat přes wsHttpBinding v čase</i>	91
<i>Obr. 15: Průběh stahování dat přes netTcpBinding v čase</i>	91
<i>Obr. 16: Průběh nahrávání dat přes WebAPI v čase</i>	91
<i>Obr. 17: Průběh nahrávání dat přes basicHttpBinding v čase</i>	92
<i>Obr. 18: Průběh nahrávání dat přes netTcpBinding v čase</i>	92
<i>Obr. 19: Průměrná režie při volání funkce dle způsobu přenosu</i>	98
<i>Obr. 20: Množství přenesených dat při volání funkce</i>	100

SEZNAM GRAFŮ

<i>Graf 1: Serializace objektu s 1 proměnnou - 16 kbit/s</i>	81
<i>Graf 2: Serializace objektu s 40 proměnnými - 16 kbit/s</i>	81
<i>Graf 3: Stahování 500B binárních dat – 16 kbit/s (vybraná hodnota)</i>	83
<i>Graf 4: Stahování 50 kB binárních dat – 128 kbit/s (vybraná hodnota)</i>	85
<i>Graf 5: Stahování 5 MB binárních dat – 10 Mbit/s (vybraná hodnota)</i>	87
<i>Graf 6: Nahrávání 500 kB binárních dat – 10 Mbit/s (vybraná hodnota)</i>	89
<i>Graf 7: Nahrávání 10 MB binárních dat – 10 Mbit/s (vybraná hodnota)</i>	89
<i>Graf 8: Stahování 1 MB binárních dat - veřejný test (vybraná hodnota)</i>	94
<i>Graf 9: Nahrávání 1 MB binárních dat - veřejný test (vybraná data)</i>	95
<i>Graf 10: Průměrná režie při volání funkce dle způsobu přenosu</i>	99
<i>Graf 11: Množství přenesených dat při volání funkce</i>	100

SEZNAM TABULEK

<i>Tab. 1: Seznam datových jednotek pro směr ke klientovi – laboratorní test</i>	<i>65</i>
<i>Tab. 2: Seznam způsobů přenosu dat pro směr ke klientovi – laboratorní test</i>	<i>66</i>
<i>Tab. 3: Seznam datových jednotek pro směr od klienta – laboratorní test.....</i>	<i>67</i>
<i>Tab. 4: Seznam způsobů přenosu dat pro směr od klienta – laboratorní test</i>	<i>67</i>
<i>Tab. 5: Seznam datových jednotek pro směr ke klientovi – veřejný test.....</i>	<i>68</i>
<i>Tab. 6: Seznam způsobů přenosu dat pro směr ke klientovi – veřejný test.....</i>	<i>68</i>
<i>Tab. 7: Seznam datových jednotek pro směr od klienta – veřejný test</i>	<i>69</i>
<i>Tab. 8: Seznam způsobů přenosu dat pro směr od klienta - veřejný test.....</i>	<i>69</i>
<i>Tab. 9: Serializace objektu – stahování 16 kbit/s</i>	<i>80</i>
<i>Tab. 10: Stahování binárních dat – část 1/3 – 16 kbit/s</i>	<i>82</i>
<i>Tab. 11: Stahování binárních dat – část 2/3 – 128 kbit/s</i>	<i>84</i>
<i>Tab. 12: Stahování binárních dat – část 3/3 – 10 Mbit/s</i>	<i>86</i>
<i>Tab. 13: Nahrávání binárních dat – 10 Mbit/s</i>	<i>88</i>
<i>Tab. 14: Stahování binárních dat – veřejný test</i>	<i>93</i>
<i>Tab. 15: Nahrávání binárních dat – veřejný test.....</i>	<i>95</i>

SEZNAM PŘÍLOH

- P I Příklad SOAP zprávy s optimalizací MTOM.
- P II Příklad popisu webové služby pomocí jazyka WSDL.
- P III Příklad přenášené zprávy při volání funkce, která vrací objekt se 40 proměnnými.
- P IV Ukázka šifrování na transportní vrstvě (TLS).
- P V CD s textem DP a aplikací použitou pro laboratorní a veřejné testování.

PŘÍLOHA P I : PŘÍKLAD SOAP ZPRÁVY S OPTIMALIZACÍ MTOM

```
Content-Type:multipart/related;
boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812;

type="application/xop+xml";
start="<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>";
start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
    <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
        <soapenv:Header/>
        <soapenv:Body>
            <ati:sendImage xmlns="http://org.apache.axis2/jaxws/sample/mtom"
xmlns:ati="http://org.apache.axis2/jaxws/sample/mtom">
                <ati:input>
                    <ati:imageData>
                        <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
                    </ati:imageData>
                </ati:input>
            </ati:sendImage>
        </soapenv:Body>
    </soapenv:Envelope>

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
    <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>

... následují binární data ...

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

PŘÍLOHA P II : PŘÍKLAD POPISU WEBOVÉ SLUŽBY POMOCÍ JAZYKA WSDL

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>
```

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

<service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>

</definitions>
```

PŘÍLOHA P III: UKÁZKY PŘENÁŠENÉ ZPRÁVY PŘI VOLÁNÍ FUNKCE, KTERÁ VRACÍ OBJEKT S 40 PROMĚNNÝMI

Ukázky zpráv zašifrovaných přes TLS jsou autentické, zpráva bude ovšem pokaždé vypadat jinak v závislosti na dohodnuté symetrické šifře a klíči. Netisknutelné binární znaky jsou nahrazeny „“. Z dat nelze nic vyčíst a slouží pro názornou demonstraci TLS.

WebAPI - JSON

```
GET /api/Test/Return40 HTTP/1.1
```

```
Accept: application/json
```

```
Host: 192.168.50.111:63282
```

```
HTTP/1.1 200 OK
```

```
Cache-Control: no-cache
```

```
Pragma: no-cache
```

```
Content-Type: application/json; charset=utf-8
```

```
Expires: -1
```

```
Server: Microsoft-IIS/8.0
```

```
X-AspNet-Version: 4.0.30319
```

```
X-Powered-By: ASP.NET
```

```
Date: Tue, 23 Apr 2013 14:14:32 GMT
```

```
Content-Length: 513
```

```
{ "Data2": "A", "Data3": "B", "Data4": "C", "Data5": "D", "Data6": "E", "Data7": "F", "Data8": "G", "Data9": "H", "Data10": "I", "Data11": "J", "Data12": "K", "Data13": "L", "Data14": "M", "Data15": "N", "Data16": "O", "Data17": "P", "Data18": "Q", "Data19": "R", "Data20": "S", "Data21": "T", "Data22": "U", "Data23": "V", "Data24": "W", "Data25": "X", "Data26": "Y", "Data27": "Z", "Data28": "1", "Data29": "2", "Data30": "3", "Data31": "4", "Data32": "5", "Data33": "6", "Data34": "7", "Data35": "8", "Data36": "9", "Data37": "0", "Data38": "-", "Data39": ".", "Data40": "{}", "Data1": "-" }
```

WebAPI - XML

```
GET /api/Test/Return40 HTTP/1.1
```

```
Accept: text/xml
```

```
Host: 192.168.50.111:63282
```

```
HTTP/1.1 200 OK
```

```
Cache-Control: no-cache
```

```
Pragma: no-cache
```

```
Content-Type: text/xml; charset=utf-8
```

```
Expires: -1
```

```
Server: Microsoft-IIS/8.0
```

```
X-AspNet-Version: 4.0.30319
```

```
X-Powered-By: ASP.NET
```

```
Date: Tue, 23 Apr 2013 14:14:32 GMT
```

```
Content-Length: 852
```

```
<ReturnFormat40 xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/TestApplication"><Data1>-
</Data1><Data10>I</Data10><Data11>J</Data11><Data12>K</Data12><Data13>L</Data13><Data14>M</
Data14><Data15>N</Data15><Data16>O</Data16><Data17>P</Data17><Data18>Q</Data18><Data19>R</D
ata19><Data2>A</Data2><Data20>S</Data20><Data21>T</Data21><Data22>U</Data22><Data23>V</Data
23><Data24>W</Data24><Data25>X</Data25><Data26>Y</Data26><Data27>Z</Data27><Data28>1</Data2
8><Data29>2</Data29><Data3>B</Data3><Data30>3</Data30><Data31>4</Data31><Data32>5</Data32><
Data33>6</Data33><Data34>7</Data34><Data35>8</Data35><Data36>9</Data36><Data37>0</Data37><D
ata38>-
</Data38><Data39>.</Data39><Data4>C</Data4><Data40>{}</Data40><Data5>D</Data5><Data6>E</Dat
a6><Data7>F</Data7><Data8>G</Data8><Data9>H</Data9></ReturnFormat40>
```

WCF – BasicHttpBinding - default

POST /WCFTestServer/basicHttpBinding-default HTTP/1.1

Content-Type: text/xml; charset=utf-8

SOAPAction: "http://tempuri.org/IWCFTest/Return40"

Host: 192.168.50.111:8733

Content-Length: 133

Expect: 100-continue

Accept-Encoding: gzip, deflate

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"><s:Body><Return40
xmlns="http://tempuri.org/"></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 1378

Content-Type: text/xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:39 GMT

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"><s:Header><ActivityId
CorrelationId="0374f3a2-5b43-4032-8c3e-3a61767394cb"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">a5a05220-ab3c-418f-
a769-31931c6a5afe</ActivityId></s:Header><s:Body><Return40Response
xmlns="http://tempuri.org/"><Return40Result
xmlns:a="http://schemas.datacontract.org/2004/07/TestApplication"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><a:Data1>-
</a:Data1><a:Data10>I</a:Data10><a:Data11>J</a:Data11><a:Data12>K</a:Data12><a:Data13>L</a:
Data13><a:Data14>M</a:Data14><a:Data15>N</a:Data15><a:Data16>O</a:Data16><a:Data17>P</a:Dat
a17><a:Data18>Q</a:Data18><a:Data19>R</a:Data19><a:Data20>S</a:Data20><
a:Data21>T</a:Data21><a:Data22>U</a:Data22><a:Data23>V</a:Data23><a:Data24>W</a:Data24><a:D
ata25>X</a:Data25><a:Data26>Y</a:Data26><a:Data27>Z</a:Data27><a:Data28>1</a:Data28><a:Data
29>2</a:Data29><a:Data30>3</a:Data30><a:Data31>4</a:Data31><a:Data32>5<
/a:Data32><a:Data33>6</a:Data33><a:Data34>7</a:Data34><a:Data35>8</a:Data35><a:Data36>9</a:
Data36><a:Data37>0</a:Data37><a:Data38>-
</a:Data38><a:Data39>.</a:Data39><a:Data40>{</a:Data40><a:Data5>D</a:D
ata5><a:Data6>E</a:Data6><a:Data7>F</a:Data7><a:Data8>G</a:Data8><a:Data9>H</a:Data9></Retu
rn40Result></Return40Response></s:Body></s:Envelope>
```

WCF – BasicHttpBinding – transportSecurity

```
.....OS....'o.<-(..p@e.....\,5.....D;q.....S.....b.....(/m....U..K.-
$.z:.....N).....3.e.n
.....|r.&K....}.....h.../u.....+Y+.d.V...X.ef..Q....1...XG.U....V...pT9.?4.O.s.
l.H.f.....&n..0.....W(x.m....]T...U....Sf.?3.u....8....W.)P4zp"...(W#C
2p....[...f.#...!dOs.q.pp...kD/.r.....0j.."L..K..o...S.D.JSp...Ht..n.9r/.?..=1...^..
.....0.O.s....C.;I..u..q.96....mt<&M.....x.B.....34..r.XB>:.m.6.`.)W.5R.c~a...+9....
".o.A.....?Ec.U5rrYN~{.zS..^.....nL.....C.T{EY..x6.w....cV.#.....#Co..2x4..K
mnx...Z.,f=.....g.....:h.V...Au..R...^.....p.?,#%.okq.y.FD.Km.....N.....8.
...w..Z*n.5....<...!.....U..@ie.....j...u..B..)....$.#...#...v.$.\U...E5k.z.#..2...)=
...7.o._.iqL.,m;?s.u...t...e.9.Yy...{s.3...0./}.....Yixw.mn6.+C.7.a.c.<U...(*O).
I.N..y...Oav.rI[Si...97k...<s.j.e.R)P.G.}T0.eE....<*.Q....L.f...:V.....V.lCm.*.
.q.5..f...Y.....3....a!M8...*Uk...4.*.e..n.pT.N.....u...C.\.....jt...q.R.[D....."Q]/..W.K
.!...~.V.O.....6..jx..*...p.E*.{.c]%.Ym4..x;B.....A.....@...7q...|h.....5.Q...TNo..
d..("6JK.=.uX;]...DY...6.U.f..A..D.`d.P.z"...S.T....h[.....%.<...U....)0W....
V...].jd...?..'s.P...].ff./6...4Fo.j.9.k.=.L.n...@...2..\"Sr`EY#i..G.....^.....^
.yt+k...r.Ol...{.....s.....m...y.....x.G.@4!.....A...4....qq.n.....{.....U.....mo7..2.
....g.I.2.*(...GG4..AG...T...J...Q.@...9.f.nR9..5..fs.....s...V.%I.E...4..<4...
..\y.W.....A.(mBW....BC...<cp2...(......L...8t.3...r79v...'Hp...<...a
y.MZ.....n<...>...VG...T..(b.8....?3e.+t9...hr]!.@!...l,)..X.....h.vc.....0...?8...<
~.e>J...l.L.\,B...`2...w.{d...<.u..+,\,R.$.$.)t.....T3...w.....4X.....\
.....Y?.....n..~..}O.o..f...T.G.....^..L.9..|9w.y.fp'....&A.....+.....E{.
.R...?>.&D..4.]C...>B...3...a..iA.Jf.d.Oh.....^..Lx.E..L.....x...p.'...y...je$.....
.....L.L.F...3.G.....sG<.....&.....^..b.)g)@?.$*R...>..Z.j\...?\...$.;MG...x.KvA.
.....[...]\`3..D.....xn.....7g.O.....2[.M.....)%.P..Z.q<?...F9.....+...%8..1...NO..
.a0....
```

WCF –WsHttpBinding – default

POST /WCFTestServer/wsHttpBinding-default HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8744

Content-Length: 4097

Expect: 100-continue

Accept-Encoding: gzip, deflate

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing" xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"><s:Header><a:Action
s:mustUnderstand="1"
u:Id="_1">http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel</a:Action><a:MessageID
u:Id="_2">urn:uuid:b281f4e5-7b09-452e-8a52-4bc854788893</a:MessageID><a:ReplyTo
u:Id="_3"><a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address></a:ReplyTo>
<a:To s:mustUnderstand="1"
u:Id="4">http://192.168.50.111:8744/WCFTestServer/wsHttpBinding-default</a:To><o:Security
s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"><u:Timestamp u:Id="uuid-2cdf77a7-586c-4d8c-b4c2-8f6251b45ffc-
32"><u:Created>2013-04-23T14:14:55.532Z</u:Created><u:Expires>2013-04-
23T14:19:55.532Z</u:Expires></u:Timestamp><c:SecurityContextToken u:Id="uuid-789864bc-f05f-
450b-a2e9-60c3ab7e4c4a-1"
xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"><c:Identifier>urn:uuid:6d2dd1c1-7af6-
4bbf-8154-2b3439648b59</c:Identifier></c:SecurityContextToken><c:DerivedKeyToken
u:Id="uuid-2cdf77a7-586c-4d8c-b4c2-8f6251b45ffc-4"
xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"><o:SecurityTokenReference><o:Reference
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/sct" URI="#uuid-789864bc-f05f-450b-
a2e9-60c3ab7e4c4a-
1"/></o:SecurityTokenReference><c:Offset>0</c:Offset><c:Length>24</c:Length><c:Nonce>FlcQC2
pC9bariNAZ5u9x0w==</c:Nonce></c:DerivedKeyToken><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/><Reference
URI="#0"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>vWSN+qVaaFisUtnIhPrOTFNxdQ
w=</DigestValue></Reference><Reference URI="#1"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>GA0icwQsAzfUOq7GylCg8CTTk6
o=</DigestValue></Reference><Reference URI="#2"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>+yWlBXTgetrHQZArEHfdsZ/9ej
o=</DigestValue></Reference><Reference URI="#3"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>o3ibE52LCPwycD7dwAsKtJa+WM
w=</DigestValue></Reference><Reference URI="#4"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>fwCPU76dagEbZVSYFbvMdJH/RP
Y=</DigestValue></Reference><Reference URI="#uuid-2cdf77a7-586c-4d8c-b4c2-8f6251b45ffc-
32"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>lgSlrw6GrLpwwI5XQn/b3EbnYG
I=</DigestValue></Reference></SignedInfo><SignatureValue>xukcoa4Tp4yjU+/w2TFcie4Tegw=</Sign
atureValue><KeyInfo><o:SecurityTokenReference><o:Reference
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk" URI="#uuid-2cdf77a7-586c-4d8c-b4c2-
8f6251b45ffc-
4"/></o:SecurityTokenReference></KeyInfo></Signature></o:Security></s:Header><s:Body
u:Id="0"><t:RequestSecurityToken
xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust"><t:RequestType>http://schemas.xmlsoap
.org/ws/2005/02/trust/Cancel</t:RequestType><t:CancelTarget><o:SecurityTokenReference
xmlns:o="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"><o:Reference URI="urn:uuid:6d2dd1c1-7af6-4bbf-8154-2b3439648b59"
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/sct"></o:Reference></o:SecurityTokenRef
erence></t:CancelTarget></t:RequestSecurityToken></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 3040

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:41 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing" xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"><s:Header><a:Action
s:mustUnderstand="1"
u:Id=" 1">http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Cancel</a:Action><a:RelatesT
o u:Id=" 2">urn:uuid:b281f4e5-7b09-452e-8a52-4bc854788893</a:RelatesTo><ActivityId
CorrelationId="c04851b5-cc2c-4f56-9262-1e2885718713"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">b7835550-c030-4253-
995d-eb1dddcf09bb</ActivityId><o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"><u:Timestamp u:Id="uuid-
789864bc-f05f-450b-a2e9-60c3ab7e4c4a-32"><u:Created>2013-04-
23T14:14:41.703Z</u:Created><u:Expires>2013-04-
23T14:19:41.703Z</u:Expires></u:Timestamp><c:DerivedKeyToken u:Id="uuid-789864bc-f05f-450b-
a2e9-60c3ab7e4c4a-4"
xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"><o:SecurityTokenReference><o:Reference
URI="urn:uuid:6d2dd1c1-7af6-4bbf-8154-2b3439648b59"
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/sct"/></o:SecurityTokenReference><c:Off
set>0</c:Offset><c:Length>24</c:Length><c:Nonce>BrdWF7eR0cuoYEXVlvsJpQ=</c:Nonce></c:Deriv
edKeyToken><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-cl4n#"><SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/><Reference
URI="# 0"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
cl4n#" /></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>BXF6YVgC7GaFcATwLFL6EsPDNM
k=</DigestValue></Reference><Reference URI="# 1"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-cl4n#" /></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>Co+CdLM6yrXJYKmiLC3jm2MbK1
E=</DigestValue></Reference><Reference URI="# 2"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-cl4n#" /></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>CFIhuov8gEcgowlidsnPzjMrAY
Y=</DigestValue></Reference><Reference URI="#uuid-789864bc-f05f-450b-a2e9-60c3ab7e4c4a-
32"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
cl4n#" /></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>9kJrARRi6iX/XH2R990/Acvrx4
E=</DigestValue></Reference></SignedInfo><SignatureValue>Skrgr6t2wHiHRd+ICZMcF0cF9Ls=</Sign
atureValue><KeyInfo><o:SecurityTokenReference><o:Reference
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk" URI="#uuid-789864bc-f05f-450b-a2e9-
60c3ab7e4c4a-
4"/></o:SecurityTokenReference></KeyInfo></Signature></o:Security></s:Header><s:Body
u:Id=" 0"><t:RequestSecurityTokenResponse
xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust"><t:RequestedTokenCancelled></t:Reques
tedTokenCancelled></t:RequestSecurityTokenResponse></s:Body></s:Envelope>
```

WCF –WsHttpBinding – transportSecurity

```
.....Gp.Q." .O.j..mf...s@...Z...[.J..!@o.53.....G..A.....0m+.) .N.uf...D.R....B.....
.A...-.{.[.n...7.kAD.....}:...Q.H;...WF...{9.....-
.....).u.A/.l.@.....t..6q.....x..z.....n..u...d.$Gq...!l..B...k.(...TDq.Zt.k./...i".....
`lB0fp.1.
.... s"...O".....r.Th4RJ.]D....O....0....{iE.VX...!...t..ceDh.....d..F...}V.2.....'.
....@.?A...qg....~.5-U...ad$/,..z.-
Eqqb2.{P...Q....4s.^...#h.+j...M8Jx.F..|...).g.....8Cx....._...%F~:...xH....n..0... '!
...v>@.e.)7].K.P e.%..YA.r...~.&.d.`E.F.~.m.zO..(?)
.y.c?.P.....e.....X.....ZM...ZP...[...}<.e?{T.....H..W!...B...d..u..h..}...P.{...#
$NE..._f...Z..n.....5..V.H\Ggb.;ilE...@..u.) .r.'jr.>_E.....1.....*pX_Q_.....
..1(...e>...I-...iH.7....{.G....I.J..(G.x.Df..gt...!..."...Ar..U..~..t.ID.4Y2....
nb$......_...F.6.....`.....e..rBBn..T.J..4..Rk.....3..c....y.....*...".u.q...E.....
@m.|.....
..&.@.8.!...CF.0....>..8Y.50..{....._v..1.....|.....MKG..>U.....R../1:>....0E
...ng.O.*.....%d.....I....t.Ob X.....'!-
.....m..a.....6.,3.\f$. ....,ftT...q.....?...W..eV~...C@@./..5.cl.....
;+.b..!l..19.84....u...` E/...z(.a.s.i.(.?.!.A..d...e..K...\.b..Q.....S.;`..).h.&O
S....,"0...jg.j....Q...!.."4.'...v.F...&".h~(.%,,{.Y..IH.'.....B..I..R..7.....f!..
S&...$.1..k.k...../...Y..%...f)...Lg!v&.D..p.I.u
E.....%P.6.....b.....W...(...?a..?..=...). .....` \: ^+...[.p..l..U.=mT..b...Q..._..el2.....8
J.G.f..T.../..l.....T..(AVr.U..p...sG).W
..<..]A.vc..T..m...ICMbZ.S5.....e.L.....)l.....!..4....&...l...G
7.!Q.....(....]@...LX,r...B...S.\.a..Y.....D4..D...4.....<n.G3...Z.....G2....
.u.....K1.....a..Z..d;l..#..Y.j...ZK...%.8...oK+,...wrSJ%
~.`.H..w...b?....e9!.....C.MQ../!4..[...AuY;..L.....[.I...IC...XB...b.....%....c!..
.m.....N1..5.?.).t'...&.....[...Uu..E..Rk.y.Au.=.U.AJ.2.q.....n..(....BUe.....<.....
".T...<...>[.$.9f....'z..t.S...f.G.....NW.....ixN.....)rT....Yd:...J)v....Jx.
.u.s.....Sw.....%e*j .l...i.E..~.L....dl;.Pl../...a..L..~..gz.....?...\rH...-
a.....>w....+. =3|.....t.....6.?pg..A...B....._...?m.p...xX.O.F.2%.D.....%G...$..
```

```
...x..(K"V..g.NG:<9.T.q.9....eQ|.g....%.n..|..<....+.....a.e..D....e.....M.V5.....h.
.Vd:6.s.w+.....H...n...m.Y.o....+.....v...(.cl....P....O..U.fa..H.....n.....`=...
.U2...c.....;...PKN.....).....".....I..Klv..Et.Ow?...-
.....o7.n6.....4Jei.9..]..@~GD..I...J0.$..Hp....
..K...I.>....t.CH....Q.:.Y..X]..]...J.....g.)K.....4Q.....H..Y..._7-
.I...=.C...swi.D.#...lK.Y.).....?..@fW.....d...#.Oai.....=...Oy.71x..4$.e...LA...}u...t..
^).+5..h"oP8.....!{.UT=p..bL._23.-..O....D....(P.

P6.^F.....fZz!Ap...)XN.-"ViG..8d.c65.....P1.=.O.(.8...
```

WCF -WsHttpBinding - noSecurity

POST /WCFTestServer/wsHttpBinding-noSecurity HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8743

Content-Length: 540

Expect: 100-continue

Accept-Encoding: gzip, deflate

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><a:Action
s:mustUnderstand="1">http://tempuri.org/IWCFTest/Return40</a:Action><a:MessageID>urn:uuid:e
ae360-99c5-4a22-a116-
2436f2220deb</a:MessageID><a:ReplyTo><a:Address>http://www.w3.org/2005/08/addressing/anonym
ous</a:Address></a:ReplyTo><a:To
s:mustUnderstand="1">http://192.168.50.111:8743/WCFTestServer/wsHttpBinding-
noSecurity</a:To></s:Header><s:Body><Return40
xmlns="http://tempuri.org/"></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 1581

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:39 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><a:Action
s:mustUnderstand="1">http://tempuri.org/IWCFTest/Return40Response</a:Action><ActivityId
CorrelationId="c49f583e-4e7a-4344-8d13-d20bba673642"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">2d888443-fd59-47d0-
a530-ce4a51d3d464</ActivityId><a:RelatesTo>urn:uuid:ae360-99c5-4a22-a116-
2436f2220deb</a:RelatesTo></s:Header><s:Body><Return40Response
xmlns="http://tempuri.org/"><Return40Result
xmlns:b="http://schemas.datacontract.org/2004/07/TestApplication"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><b>Data1>-
</b>Data1><b>Data10>I</b>Data10><b>Data11>J</b>Data11><b>Data12>K</b>Data12><b>Data13>L</b:
Data13><b>Data14>M</b>Data14><b>Data15>N</b>Data15><b>Data16>O</b>Data16><b>Data17>P</b:Dat
a17><b>Data18>Q</b>Data18><b>Data19>R</b>Data19><b>Data20>A</b>Data20><b>Data21>S</b>Data21><
b>Data22>T</b>Data22><b>Data23>U</b>Data23><b>Data24>V</b>Data24><b>Data25>W</b>Data25><b:D
ata26>X</b>Data26><b>Data27>Y</b>Data27><b>Data28>Z</b>Data28><b>Data29>1</b>Data29><b>Data
30>2</b>Data30><b>Data31>3</b>Data31><b>Data32>4</b>Data32><b>Data33>5</b>Data33><b>Data34>6</b:
Data34><b>Data35>7</b>Data35><b>Data36>8</b>Data36><b>Data37>9</b>Data37><b>Data38>-
</b>Data38><b>Data39>.</b>Data39><b>Data40>C</b>Data40><b>Data41>{</b>Data41><b>Data42>D</b:D
ata42><b>Data43>E</b>Data43><b>Data44>F</b>Data44><b>Data45>G</b>Data45><b>Data46>H</b>Data46></Retu
rn40Result></Return40Response></s:Body></s:Envelope>
```

WCF –WsHttpBinding - reliableNoSecurity

POST /WCFTTestServer/wsHttpBinding-realiabileNoSecurity HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8739

Content-Length: 931

Expect: 100-continue

Accept-Encoding: gzip, deflate

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><r:SequenceAcknowledgement><r:Identifier><urn:uuid:cbe9d97e-4fe7-438d-9b1e-2f8a47d273ee</r:Identifier><r:AcknowledgementRange Lower="1" Upper="9"/></r:SequenceAcknowledgement><r:Sequence
s:mustUnderstand="1"><r:Identifier><urn:uuid:6decdf73-b872-4eb8-aec2-
3b65ba3650dd</r:Identifier><r:MessageNumber>10</r:MessageNumber></r:Sequence><a:Action
s:mustUnderstand="1">http://tempuri.org/IWCFTTest/Return40</a:Action><a:MessageID><urn:uuid:7
9cacb66-4020-47af-8a18-
dcef78825981</a:MessageID><a:ReplyTo><a:Address>http://www.w3.org/2005/08/addressing/anonym
ous</a:Address></a:ReplyTo><a:To
s:mustUnderstand="1">http://192.168.50.111:8739/WCFTTestServer/wsHttpBinding-
realiabileNoSecurity</a:To></s:Header><s:Body><Return40
xmlns="http://tempuri.org/"></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 2069

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:39 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><r:Sequence
s:mustUnderstand="1"><r:Identifier><urn:uuid:cbe9d97e-4fe7-438d-9b1e-
2f8a47d273ee</r:Identifier><r:MessageNumber>10</r:MessageNumber></r:Sequence><r:SequenceAck
nowledgement><r:Identifier><urn:uuid:6decdf73-b872-4eb8-aec2-
3b65ba3650dd</r:Identifier><r:AcknowledgementRange Lower="1"
Upper="10"/></r:SequenceAcknowledgement><a:Action
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining></r:Seque
nceAcknowledgement><a:Action
s:mustUnderstand="1">http://tempuri.org/IWCFTTest/Return40Response</a:Action><ActivityId
CorrelationId="ce4bc66c-c771-49fd-8ea1-bea957b307cc"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">44f4239f-e696-4110-
9dae-9895054b14f1</ActivityId><a:RelatesTo><urn:uuid:79cacb66-4020-47af-8a18-
dcef78825981</a:RelatesTo></s:Header><s:Body><Return40Response
xmlns="http://tempuri.org/"><Return40Result
xmlns:b="http://schemas.datacontract.org/2004/07/TestApplication"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><b>Data1>-
</b>Data1><b>Data10>I</b>Data10><b>Data11>J</b>Data11><b>Data12>K</b>Data12><b>Data13>L</b:
Data13><b>Data14>M</b>Data14><b>Data15>N</b>Data15><b>Data16>O</b>Data16><b>Data17>P</b:Dat
a17><b>Data18>Q</b>Data18><b>Data19>R</b>Data19><b>Data20>A</b>Data20><b>Data21>S</b>Data21><
b>Data22>T</b>Data22><b>Data23>U</b>Data23><b>Data24>V</b>Data24><b>Data25>W</b>Data25><b:
Data26>X</b>Data26><b>Data27>Y</b>Data27><b>Data28>Z</b>Data28><b>Data29>1</b>Data29><b:Data
30>2</b>Data30><b>Data31>3</b>Data31><b>Data32>4</b>Data32><b>Data33>5</b>Data33><b>Data34>6</
b>Data34><b>Data35>7</b>Data35><b>Data36>8</b>Data36><b>Data37>9</b>Data37><b>Data38>-
</b>Data38><b>Data39>.</b>Data39><b>Data40>C</b>Data40><b>Data41>{</b>Data41><b>Data42>D</b:D
ata42><b>Data43>E</b>Data43><b>Data44>F</b>Data44><b>Data45>G</b>Data45><b>Data46>H</b>Data46></Retu
rn40Result></Return40Response></s:Body></s:Envelope>
```

Příklad komunikace týkající se kanálu (započetí spojení):

POST /WCFTTestServer/wsHttpBinding-reliableNoSecurity HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8739

Content-Length: 690

Expect: 100-continue

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence</a:Action><a:MessageID>urn:uuid:5d070b18-688d-44bb-b06b-d98431549c2f</a:MessageID><a:To
s:mustUnderstand="1">http://192.168.50.111:8739/WCFTTestServer/wsHttpBinding-
reliableNoSecurity</a:To></s:Header><s:Body><CreateSequence
xmlns="http://schemas.xmlsoap.org/ws/2005/02/rm"><AcksTo><a:Address>http://www.w3.org/2005/
08/addressing/anonymous</a:Address></AcksTo><Offer><Identifier>urn:uuid:cbe9d97e-4fe7-438d-
9b1e-2f8a47d273ee</Identifier></Offer></CreateSequence></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 820

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:36 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequenceResponse</a:Act
ion><a:RelatesTo>urn:uuid:5d070b18-688d-44bb-b06b-d98431549c2f</a:RelatesTo><ActivityId
CorrelationId="ee9bd3d3-9d69-4581-ab03-8788196d6de2"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">00000000-0000-0000-
0200-0080010000e6</ActivityId></s:Header><s:Body><CreateSequenceResponse
xmlns="http://schemas.xmlsoap.org/ws/2005/02/rm"><Identifier>urn:uuid:6decdf73-b872-4eb8-
aec2-
3b65ba3650dd</Identifier><Accept><AcksTo><a:Address>http://192.168.50.111:8739/WCFTTestServe
r/wsHttpBinding-
reliableNoSecurity</a:Address></AcksTo></Accept></CreateSequenceResponse></s:Body></s:Enve
lope>
```

Příklad komunikace týkající se kanálu (ukončení spojení):

POST /WCFTTestServer/wsHttpBinding-reliableNoSecurity HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8739

Content-Length: 753

Expect: 100-continue

Accept-Encoding: gzip, deflate

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><r:SequenceAcknowledgement><r:Iden
tifier>urn:uuid:cbe9d97e-4fe7-438d-9b1e-2f8a47d273ee</r:Identifier><r:AcknowledgementRange
Lower="1" Upper="10"/></r:SequenceAcknowledgement><r:Sequence
s:mustUnderstand="1"><r:Identifier>urn:uuid:6decdf73-b872-4eb8-aec2-
3b65ba3650dd</r:Identifier><r:MessageNumber>11</r:MessageNumber><r:LastMessage/></r:Sequenc
e><a:Action
```

s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/LastMessage</a:Action><a:To s:mustUnderstand="1">http://192.168.50.111:8739/WCFTestServer/wsHttpBinding-reliableNoSecurity</a:To></s:Header><s:Body/></s:Envelope>

HTTP/1.1 200 OK

Content-Length: 750

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:41 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><r:Sequence
s:mustUnderstand="1"><r:Identifier>urn:uuid:cbe9d97e-4fe7-438d-9b1e-
2f8a47d273ee</r:Identifier><r:MessageNumber>11</r:MessageNumber><r:LastMessage/></r:Sequenc
e><r:SequenceAcknowledgement><r:Identifier>urn:uuid:6decdf73-b872-4eb8-aec2-
3b65ba3650dd</r:Identifier><r:AcknowledgementRange Lower="1"
Upper="11"/><netrm:BufferRemaining
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining></r:Seque
nceAcknowledgement><a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/LastMessage</a:Action></s:Hea
der><s:Body/></s:Envelope>
```

POST /WCFTestServer/wsHttpBinding-reliableNoSecurity HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8739

Content-Length: 783

Expect: 100-continue

Accept-Encoding: gzip, deflate

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><r:SequenceAcknowledgement><r:Iden
tifier>urn:uuid:cbe9d97e-4fe7-438d-9b1e-2f8a47d273ee</r:Identifier><r:AcknowledgementRange
Lower="1" Upper="11"/></r:SequenceAcknowledgement><a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/TerminateSequence</a:Action><
a:MessageID>urn:uuid:f0df6b89-ba7d-4daf-9fed-72445b9ffd49</a:MessageID><a:To
s:mustUnderstand="1">http://192.168.50.111:8739/WCFTestServer/wsHttpBinding-
reliableNoSecurity</a:To></s:Header><s:Body><r:TerminateSequence><r:Identifier>urn:uuid:6d
ecdf73-b872-4eb8-aec2-
3b65ba3650dd</r:Identifier></r:TerminateSequence></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 708

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:41 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><r:SequenceAcknowledgement><r:Iden
tifier>urn:uuid:6decdf73-b872-4eb8-aec2-3b65ba3650dd</r:Identifier><r:AcknowledgementRange
Lower="1" Upper="11"/><netrm:BufferRemaining
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining></r:Seque
nceAcknowledgement><a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/TerminateSequence</a:Action><
/s:Header><s:Body><r:TerminateSequence><r:Identifier>urn:uuid:cbe9d97e-4fe7-438d-9b1e-
2f8a47d273ee</r:Identifier></r:TerminateSequence></s:Body></s:Envelope>
```

WCF –WsHttpBinding - defaultNoSecurity

POST /WCFTTestServer/wsHttpBinding-defaultWithReliable HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8

Host: 192.168.50.111:8749

Content-Length: 4267

Expect: 100-continue

Accept-Encoding: gzip, deflate

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing" xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"><s:Header><r:SequenceAcknowledgement u:Id=" 0"><r:Identifier>urn:uuid:689d18ba-
53e6-4788-8ae8-35a437f60374</r:Identifier><r:AcknowledgementRange Lower="1"
Upper="9"/></r:SequenceAcknowledgement><r:Sequence s:mustUnderstand="1"
u:Id=" 1"><r:Identifier>urn:uuid:6356198a-3292-460e-a4ba-
87567012f19e</r:Identifier><r:MessageNumber>10</r:MessageNumber></r:Sequence><a:Action
s:mustUnderstand="1" u:Id=" 2">http://tempuri.org/IWCFTTest/Return40</a:Action><a:MessageID
u:Id=" 3">urn:uuid:389282e9-07da-4fef-ad2a-2c76a7a4e489</a:MessageID><a:ReplyTo
u:Id=" 4"><a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address></a:ReplyTo>
<a:To s:mustUnderstand="1"
u:Id=" 5">http://192.168.50.111:8749/WCFTTestServer/wsHttpBinding-
defaultWithReliable</a:To><o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"><u:Timestamp u:Id="uuid-
2cdf77a7-586c-4d8c-b4c2-8f6251b45ffc-31"><u:Created>2013-04-
23T14:14:53.443Z</u:Created><u:Expires>2013-04-
23T14:19:53.443Z</u:Expires></u:Timestamp><c:SecurityContextToken u:Id="uuid-789864bc-f05f-
450b-a2e9-60c3ab7e4c4a-11"
xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"><c:Identifier>urn:uuid:632bc233-1cc3-
4d8a-b83a-4e2889d18016</c:Identifier><c:SecurityContextToken><c:DerivedKeyToken
u:Id="uuid-2cdf77a7-586c-4d8c-b4c2-8f6251b45ffc-14"
xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"><o:SecurityTokenReference><o:Reference
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/sct" URI="#uuid-789864bc-f05f-450b-
a2e9-60c3ab7e4c4a-
11"/></o:SecurityTokenReference><c:Offset>0</c:Offset><c:Length>24</c:Length><c:Nonce>GkBLW
NXGwd6bNdwR38mmLw==</c:Nonce></c:DerivedKeyToken><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-shal"/><Reference
URI="# 0"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>F9T1KNFg7gN9E+H/zBdwJDI52a
g=</DigestValue></Reference><Reference URI="# 1"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>6tQGGu/jPZaYn25KiaZZBrkLYP
M=</DigestValue></Reference><Reference URI="# 2"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>nmFISr+PikeBn1MuOMGVcIGUNB
o=</DigestValue></Reference><Reference URI="# 3"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>lJ4qeWCxtulxEHJUMbjHLzmDwj
o=</DigestValue></Reference><Reference URI="# 4"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>l6mMmQ2LE9VftjaA6Qc4GKBXUR
w=</DigestValue></Reference><Reference URI="# 5"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>695urD67TaPpQ/m28Jwec6zAK/
A=</DigestValue></Reference><Reference URI="#uuid-2cdf77a7-586c-4d8c-b4c2-8f6251b45ffc-
31"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>UK4IE+rIMyFz+hrsTfyQHTTlxs
U=</DigestValue></Reference></SignedInfo><SignatureValue>kKp602TbSfPcu3DiHnTyQE86Dj4=</Sign
atureValue><KeyInfo><o:SecurityTokenReference><o:Reference
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk" URI="#uuid-2cdf77a7-586c-4d8c-b4c2-
8f6251b45ffc-
14"/></o:SecurityTokenReference></KeyInfo></Signature></o:Security></s:Header><s:Body><Retu
rn40 xmlns="http://tempuri.org/"></s:Body></s:Envelope>
```

HTTP/1.1 200 OK

Content-Length: 4673

Content-Type: application/soap+xml; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Tue, 23 Apr 2013 14:14:39 GMT

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing" xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"><s:Header><r:Sequence
s:mustUnderstand="1" u:Id="_0"><r:Identifier>urn:uuid:689d18ba-53e6-4788-8ae8-
35a437f60374</r:Identifier><r:MessageNumber>10</r:MessageNumber><r:SequenceAck
nowledgement u:Id="_1"><r:Identifier>urn:uuid:6356198a-3292-460e-a4ba-
87567012f19e</r:Identifier><r:AcknowledgementRange Lower="1"
Upper="10"/><netrm:BufferRemaining
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining></r:Seque
nceAcknowledgement><a:Action s:mustUnderstand="1"
u:Id="_2">http://tempuri.org/IWCFTTest/Return40Response</a:Action><ActivityId
CorrelationId="89c9b701-7de1-45ed-86e6-99bd5c1e30f9"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">030c12d5-5291-4655-
a9f5-76d2a76d15c8</ActivityId><a:RelatesTo u:Id="_3">urn:uuid:389282e9-07da-4fef-ad2a-
2c76a7a4e489</a:RelatesTo><o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"><u:Timestamp u:Id="uid-
789864bc-f05f-450b-a2e9-60c3ab7e4c4a-31"><u:Created>2013-04-
23T14:14:39.352Z</u:Created><u:Expires>2013-04-
23T14:19:39.352Z</u:Expires></u:Timestamp><c:DerivedKeyToken u:Id="uid-789864bc-f05f-450b-
a2e9-60c3ab7e4c4a-14"
xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"><o:SecurityTokenReference><o:Reference
URI="urn:uuid:632bc233-1cc3-4d8a-b83a-4e2889d18016"
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/sct"/></o:SecurityTokenReference><c:Off
set>0</c:Offset><c:Length>24</c:Length><c:Nonce>mlqHfquYAzZFi3cIbXft+g==</c:Nonce></c:Deriv
edKeyToken><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/></Reference
URI="# 0"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>1EktW8eSe2oxb90h5IXHscbQZx
I=</DigestValue></Reference><Reference URI="# 1"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>xpT/Tis7bflF3IVgz6xM504knT
0=</DigestValue></Reference><Reference URI="# 2"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>jdSiIPqwHExqcY3GU1zN8dNNkf
k=</DigestValue></Reference><Reference URI="# 3"><Transforms><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>deI9nESWvdcHmQANyT7nNfyALX
g=</DigestValue></Reference><Reference URI="#uid-789864bc-f05f-450b-a2e9-60c3ab7e4c4a-
31"><Transforms><Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></Transforms><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><DigestValue>z8gvaD5//C30TguiWOYMDhbb9H
I=</DigestValue></Reference></SignedInfo><SignatureValue>L2robVv9CfOV4MZAgolM0sC/Yoo=</Sign
atureValue><KeyInfo><o:SecurityTokenReference><o:Reference
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk" URI="#uid-789864bc-f05f-450b-a2e9-
60c3ab7e4c4a-
14"/></o:SecurityTokenReference></KeyInfo></Signature></o:Security></s:Header><s:Body><Retu
rn40Response xmlns="http://tempuri.org/"><Return40Result
xmlns:b="http://schemas.datacontract.org/2004/07/TestApplication"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><b>Data1>-
</b>Data1><b>Data10>I</b>Data10><b>Data11>J</b>Data11><b>Data12>K</b>Data12><b>Data13>L</b:
Data13><b>Data14>M</b>Data14><b>Data15>N</b>Data15><b>Data16>O</b>Data16><b>Data17>P</b:Dat
a17><b>Data18>Q</b>Data18><b>Data19>R</b>Data19><b>Data20>A</b>Data20><b>Data21>T</b:Data21><b:Data22>U</b:Data22><b:Data23>V</b:Data23><b:Data24>W</b:Data24><b:Data25>X</b:Data25><b:Data26>Y</b:Data26><b:Data27>Z</b:Data27><b:Data28>1</b:Data28><b:Data29>2</b:Data29><b:Data30>3</b:Data30><b:Data31>4</b:Data31><b:Data32>5</b:Data32><b:Data33>6</b:Data33><b:Data34>7</b:Data34><b:Data35>8</b:Data35><b:Data36>9</b:Data36><b:Data37>0</b:Data37><b:Data38>-
</b:Data38><b:Data39>.</b:Data39><b:Data40>{</b:Data40><b:Data41>}</b:Data41><b:Data42>D</b:Data42><b:Data43>E</b:Data43><b:Data44>F</b:Data44><b:Data45>G</b:Data45><b:Data46>H</b:Data46><b:Data47>I</b:Data47><b:Data48>J</b:Data48><b:Data49>K</b:Data49><b:Data50>L</b:Data50><b:Data51>M</b:Data51><b:Data52>N</b:Data52><b:Data53>O</b:Data53><b:Data54>P</b:Data54><b:Data55>Q</b:Data55><b:Data56>R</b:Data56><b:Data57>S</b:Data57><b:Data58>T</b:Data58><b:Data59>U</b:Data59><b:Data60>V</b:Data60><b:Data61>W</b:Data61><b:Data62>X</b:Data62><b:Data63>Y</b:Data63><b:Data64>Z</b:Data64><b:Data65>[</b:Data65><b:Data66>]</b:Data66><b:Data67>^</b:Data67><b:Data68>_</b:Data68><b:Data69>`</b:Data69><b:Data70>a</b:Data70><b:Data71>b</b:Data71><b:Data72>c</b:Data72><b:Data73>d</b:Data73><b:Data74>e</b:Data74><b:Data75>f</b:Data75><b:Data76>g</b:Data76><b:Data77>h</b:Data77><b:Data78>i</b:Data78><b:Data79>j</b:Data79><b:Data80>k</b:Data80><b:Data81>l</b:Data81><b:Data82>m</b:Data82><b:Data83>n</b:Data83><b:Data84>o</b:Data84><b:Data85>p</b:Data85><b:Data86>q</b:Data86><b:Data87>r</b:Data87><b:Data88>s</b:Data88><b:Data89>t</b:Data89><b:Data90>u</b:Data90><b:Data91>v</b:Data91><b:Data92>w</b:Data92><b:Data93>x</b:Data93><b:Data94>y</b:Data94><b:Data95>z</b:Data95><b:Data96>AA</b:Data96><b:Data97>BB</b:Data97><b:Data98>CC</b:Data98><b:Data99>DD</b:Data99><b:Data100>EE</b:Data100></b>Data100></Return40Result></Return40Response></s:Body></s:Envelope>
```

V kanálu jsou dále zasílány informace týkající se Reliable Session, z důvodu délky zde nebudou uvedeny.

WCF – NetTcpBinding - default

```
.....(x..J{Z.)UX;~nB..Z.g([...+vN.....+h..e=.zj.A.#x.F". .....9..}P..fY.h/'..W...T...L...
.....:2...2J...X.dJV...w.wI...Xi...&...Ng..p..05.|a]_.....j_T.,mP,.2...aL..A.,N...
.C)<.....U.h.4..;^.[.....Fb.....:3.....r.[.....;.....3...eJ+. ([.t...m...G.....9.tIibUU...
..i6.....*.....m;.....i.5I..$RDe.G..8h...n!<N...Vt..g]%4..?u2syzB1.y.O[.b./k...a...
..G.....K*.j.3s.0...=<.....%S.....s...kh.)..K\..5.L..H...G@..G5.R.N....'.n_.fj.s21..F.....
.....`;.fx..9..I.V..7$.I.D.../v..%.....P-..H/ .....7....
.e....._y.\..3f.....!,.....[.....d..._,.....a.....Xs...=,e&...k.....b.....|x7.GR.....[.R{
.W......W]q.....\.....&,..p.)].QmGG.q...T..y.4.|.e.`.z..EvE..uW.....o.....I.....<..S
.eL...*.|9.....*x.....`E,.....bq..qt...X....}.q...F.....$.N.G@.C5wS~C...r...X.{....
5%.e.Z.(.v
```

WCF – NetTcpBinding - defaultWithReliable

```
.....s].K...V.`H.....f...J...+.O.Q5.....,2..}.._..ie...../...QF..8.....b....#V...D(...ky...+
i...|.X>...C...G.^...9...j...^A...K...9..".....vj.o.$....b...A.`{...=Xh...._..b...X.H]zv
9..RT.Hh8.f.8.....pV.....e*..Tt.^RC.....1...`8F..8...Eh..e...XgW.....8,Od./|..
;.s.W..8.L|..*} ...[f..sv.n
.<..}kl1=f.....y.k....."t.@.=.bTN!c...Bz.+|.gW<...~`.p=....._..Ma9t...V..?.KUh(sAb.'..."
.:9.s.&.(:=5...*.....e/.3hU..W...=;w...Y.....'7..].=.:.:.....q{;.}..O.
.y.9].mw...'.{...}.....L.....Q.M..).<.....i...,.?./...`>.....<uY...'.{;.2...Cpn....
.....Y.m...D.....s.....%2m?...B.....-
..&.d.....G.....j...T.A.G..9.....AB#+j...{0_...c-
..YU.....W.9.....D.O..1.i.P.^v.....M.p`.T...h...b...G.t.[...T).cSw....Az.>.(.!.;f
bu...d.B.L...%.....g..a...jg.....Q.Pj.....[.P. y..B.....n.y<y...pv.c.-
...F./4...~...Q}}(.5.f...f..!.H...k^f...U...".z.....S.i^)..d.|.....v..P...
....5c.....f.0%P"....._..a.....V..E.
```

WCF – NetTcpBinding - noSecurity

```
.m.$http://tempuri.org/IWCFTest/Return40.Return40V...s...a.V.D
.....D...f..iTJK.....*'D,D*...D.....V.B.
.....,http://tempuri.org/IWCFTest/Return40Response.Return40Response.Return40Result.Data
21.Data22.Data23.Data24.Data25.Data26.Data27.Data28.Data29.Data30.Data31.Data32.Data33.Data
34.Data35.Data36.Data37.Data38.Data39.Data40V...s...a.V.D
....G@
ActivityId.
CorrelationId.$5ae60426-dfac-4175-a368-
7ecccea43ee5.=http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics.VR...{'A..6R0..
.D...f..iTJK.....*'D.....V.BI
.BK..b...i.E
...-
E#..IE3..JE5..KE7..LE9..ME;.NE=.OE?.PEA..QEC..RE...AEE..SEM..TEO..UEQ..VES..WEU..XEW..YE
Y..ZE[E]..2E...BE...3Ea..4Ec..5Ee..6Eg..7Ei..8Ek..9Em.Eo...-
Eq...E...CES...{E...DE%..EE'..FE)..GE+.H.....
```

Zde je možno si všimnout absence tagů pro proměnné Data1-Data20, je to z toho důvodu, že se při serializaci jmen proměnných používá slovníkové komprimační metody, kde se při prvním výskytu daného jména přiřadí danému jménu zástupný symbol. Další výskyty daného jména jsou nahrazeny daným symbolem. Slovník je postupně budován od začátku kanálu a je společný pro všechny zprávy. Důvodem toho, že tato zpráva obsahuje definici zástupných symbolů pouze pro proměnné Data21 až Data40 je ten, že před zasláním této zprávy následovala zpráva, kde se přenášelo pouze prvních 20 proměnných daného objektu (ze 40) a jména proměnných jsou stejná. Hodnoty proměnných se přenáší vždy.

Předchozí zpráva v rámci kanálu:

```
.m.$http://tempuri.org/IWCFTest/Return20.Return20V...s...a.V.D
....D..4....~.G..k.;...D,D*...D.....V.B.
.....,http://tempuri.org/IWCFTest/Return20Response.Return20Response.Return20Result.Data
11.Data12.Data13.Data14.Data15.Data16.Data17.Data18.Data19.Data20V...s...a.V.D
....-@
ActivityId.
CorrelationId.$003b7b53-4d71-43dc-98eb-
ae16a0a6b5bd.=http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics..\Tm.Y.G.M....
"D..4....~.G..k.;...D.....V.B/
.B1..b...i.E
..-
E#..IE3..JE5..KE7..LE9..ME;..NE=..OE?..PEA..QEC..RE...AEE..SE...BE...CE...DE%..EE'..FE)..GE
+..H....
```

Zpráva přenáší objekt s 20 proměnnými, současně definuje zástupné symboly pro jména proměnných, které dosud nebyly kanálem zaslány.

WCF – customBinding – přenos přes HTTP, binární serializace

```
POST //WCFTestServer/basicHttpBinary HTTP/1.1
Content-Type: application/soap+msbinl
Host: 192.168.50.111:8761
Content-Length: 182
Expect: 100-continue
Accept-Encoding: gzip, deflate

HTTP/1.1 100 Continue

V...s...a.V.D...$http://tempuri.org/IWCFTest/Return40D...,....DN..5....7D,D*...D.....9http
://192.168.50.111:8761//WCFTestServer/basicHttpBinary.V.@.Return40..http://tempuri.org/...

HTTP/1.1 200 OK
Content-Length: 821
Content-Type: application/soap+msbinl
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 24 Apr 2013 17:41:18 GMT

V...s...a.V.D...,http://tempuri.org/IWCFTest/Return40Response@
ActivityId.
CorrelationId.$b6ffe054-7d5c-4edd-aeac-
e253af9ef5c6.=http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics.K...F+.C.o.a..o
|D...,....DN..5....7.V.@.Return40Response..http://tempuri.org/@.Return40Result..b7http://sc
hemas.datacontract.org/2004/07/TestApplication..i)http://www.w3.org/2001/XMLSchema-
instance.Data1..-
_.Data10_.I_.Data11..J_.Data12..K_.Data13..L_.Data14..M_.Data15..N_.Data16..O_.Data17..P_.D
ata18..Q_.Data19..R_.Data20..S_.Data21..T_.Data22..U_.Data23..V_.Data24..W_.Data2
5..X_.Data26..Y_.Data27..Z_.Data28.._.Data29..2_.Data30..3_.Data31..4_.Data32..5_.
Data33..6_.Data34..7_.Data35..8_.Data36..9_.Data37.._.Data38..-
_.Data39..._.Data40..{ }_.Data5..D_.Data6..E_.Data7..F_.Data8..G_.Data9..H....
```

Serializace je velmi podobná Net.Tcp, nedochází zde ovšem k budování slovníku pro jména proměnných.

WCF – WebHttpBinding – JSON

Test/Return40 HTTP/1.1

Accept: application/json

Host: 192.168.50.111:8771

HTTP/1.1 200 OK

Content-Length: 513

Content-Type: application/json; charset=utf-8

Server: Microsoft-HTTPAPI/2.0

Date: Fri, 26 Apr 2013 17:49:56 GMT

```
{"Data1": "-  
", "Data10": "I", "Data11": "J", "Data12": "K", "Data13": "L", "Data14": "M", "Data15": "N", "Data16": "O",  
", "Data17": "P", "Data18": "Q", "Data19": "R", "Data2": "A", "Data20": "S", "Data21": "T", "Data22": "U",  
", "Data23": "V", "Data24": "W", "Data25": "X", "Data26": "Y", "Data27": "Z", "Data28": "1", "Data29": "2",  
", "Data3": "B", "Data30": "3", "Data31": "4", "Data32": "5", "Data33": "6", "Data34": "7", "Data35": "8",  
", "Data36": "9", "Data37": "0", "Data38": "-  
", "Data39": ".", "Data4": "C", "Data40": "{}", "Data5": "D", "Data6": "E", "Data7": "F", "Data8": "G", "D  
ata9": "H"}
```

WCF – WebHttpBinding – XML

GET /api/Test/Return40 HTTP/1.1

Accept: text/xml

Host: 192.168.50.111:8771

HTTP/1.1 200 OK

Content-Length: 852

Content-Type: application/xml; charset=utf-8

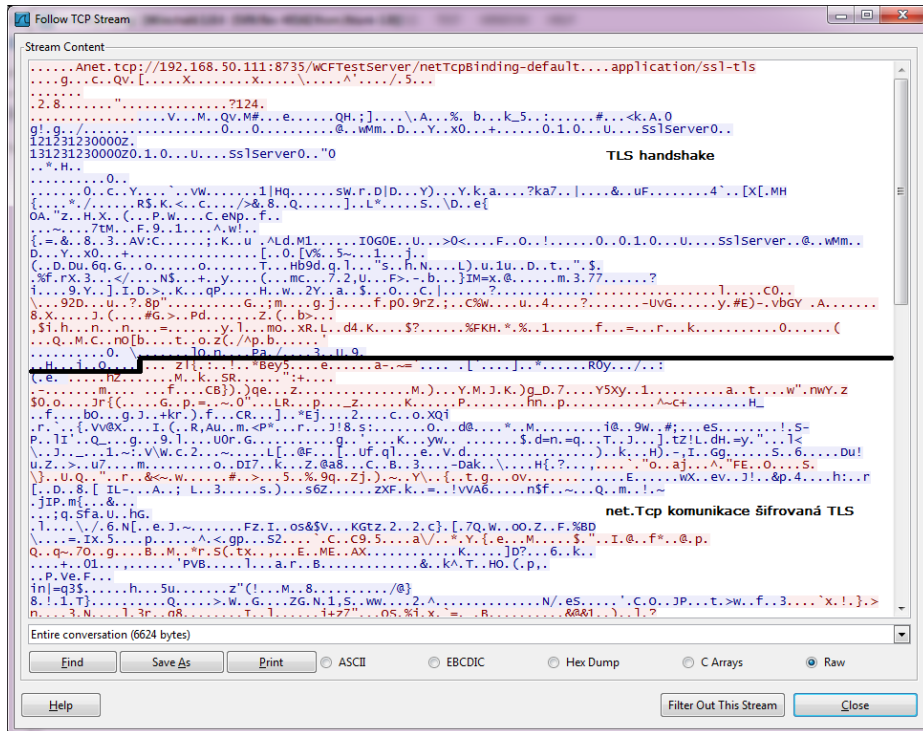
Server: Microsoft-HTTPAPI/2.0

Date: Fri, 26 Apr 2013 17:44:00 GMT

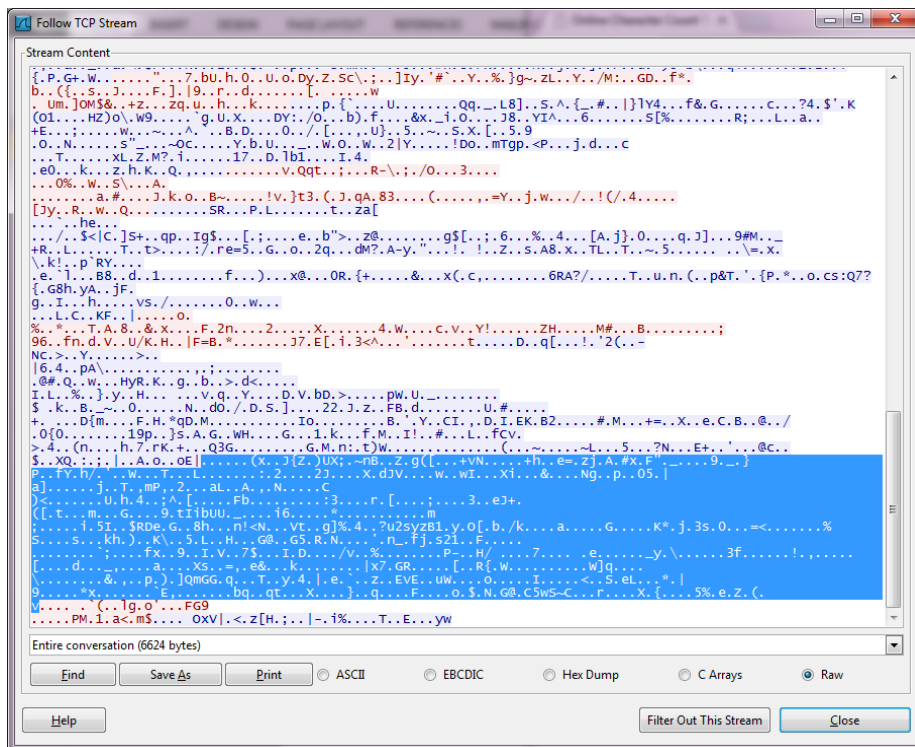
```
<ReturnFormat40 xmlns="http://schemas.datacontract.org/2004/07/TestApplication"  
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><Data1>-  
</Data1><Data10>I</Data10><Data11>J</Data11><Data12>K</Data12><Data13>L</Data13><Data14>M</  
Data14><Data15>N</Data15><Data16>O</Data16><Data17>P</Data17><Data18>Q</Data18><Data19>R</D  
ata19><Data2>A</Data2><Data20>S</Data20><Data21>T</Data21><Data22>U</Data22><Data23>V</Data  
23><Data24>W</Data24><Data25>X</Data25><Data26>Y</Data26><Data27>Z</Data27><Data28>1</Data2  
8><Data29>2</Data29><Data3>B</Data3><Data30>3</Data30><Data31>4</Data31><Data32>5</Data32><  
Data33>6</Data33><Data34>7</Data34><Data35>8</Data35><Data36>9</Data36><Data37>0</Data37><D  
ata38>-  
</Data38><Data39>.</Data39><Data4>C</Data4><Data40>{}</Data40><Data5>D</Data5><Data6>E</Dat  
a6><Data7>F</Data7><Data8>G</Data8><Data9>H</Data9></ReturnFormat40>
```

Při srovnání WebAPI a WCF WebHttpBinding má WebHttpBinding ve výchozím nastavení o 100 B nižší režii (méně položek v hlavičce).

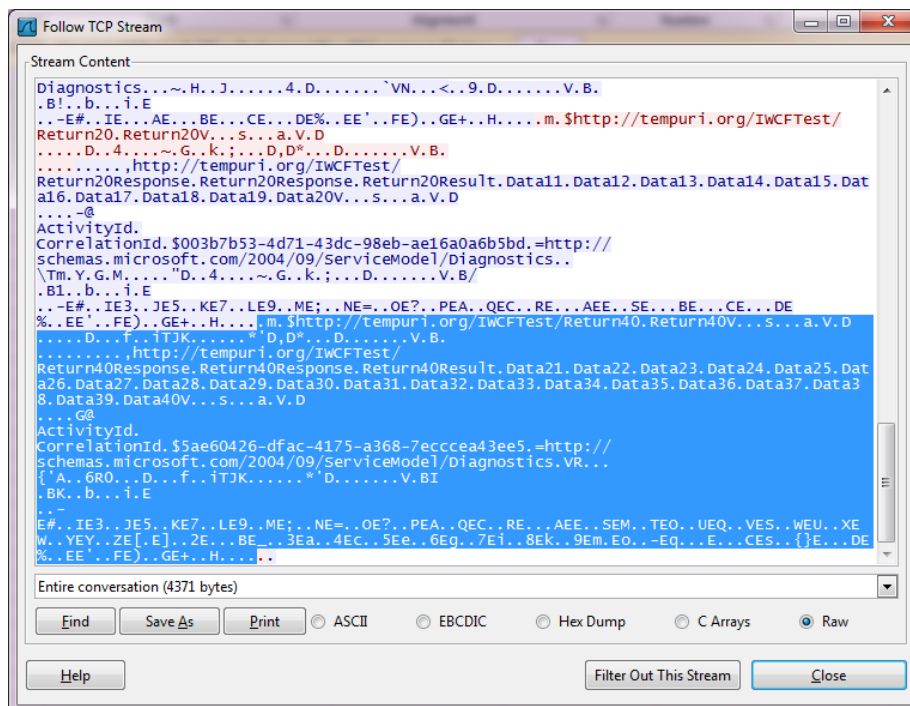
PŘÍLOHA P IV : UKÁZKA ŠIFROVÁNÍ NA TRANSPORTNÍ VRTVĚ



Ukázka 4 cestného TLS handshake.



Objekt se 40 proměnnými přenášen pomocí protokolu Net.Tcp, kanál je šifrován TLS.



Objekt se 40 proměnnými přenášen pomocí protokolu Net.Tcp,
kanál není šifrován.