VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO MONITOROVÁNÍ A ANALÝZU APLIKAČNÍHO SERVERU XTEND

XTEND MONITORING AND ANALYSIS TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ KOCMAN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. FRANTIŠEK ŠČUGLÍK, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19495/2016/xkocma04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Kocman Tomáš**
Obor: Informační technologie
Téma: **Nástroj pro monitorování a analýzu aplikačního serveru Xtend**
Xtend Monitoring and Analysis Tool
Kategorie: Počítačové sítě

Pokyny:

1. Prostudujte možnosti monitorování a analýzy datových přenosů mezi aplikacemi.
2. Seznamte se s existujícími nástroji pro monitorování a analýzu datových přenosů, např. Wireshark, Capsa Free, apod.
3. Navrhněte systém pro monitorování a analýzu aplikačního serveru Xtend.
4. Navržený systém implementujte.
5. Implementovaný systém otestujte a demonstrujte jeho funkčnost.

Literatura:

- Davidoff S., Ham J.: Network Forensics: Tracking Hackers through Cyberspace, Prentice Hall, 2012, ISBN 978-0132564717
- Sanders C.: Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems, No Starch Press, 2011, ISBN 978-1593272661

Pro udělení zápočtu za první semestr je požadováno:

- Body 1. až 3. zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Ščuglík František, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce je zaměřena na monitorování a analýzu datových přenosů mezi aplikacemi na síti a tvorbou vlastního monitorovacího nástroje, který by vyhověl požadavkům aplikačního serveru Xtend společnosti Xura, Inc. V úvodních kapitolách jsou popsány teoretické znalosti z oblasti zachytávání a analýzy dat a samotný aplikační server Xtend spolu se službami, které poskytuje. Práce je zaměřena především na implementaci systému, který spolupracuje se serverem Xtend a dokáže jeho výstup dále zpracovat a interpretovat.

Abstract

This thesis is focused on the monitoring and analysis of data transfer between applications on the network and creating custom monitoring tool that would meet the requirements of the application server Xtend of company Xura, Inc. The introductory chapters describe theoretical background of data capture and analysis, and application server Xtend with the services it provides. The thesis is mainly focused on the implementation of a system that cooperates with the server Xtend and its output can be further processed and interpreted.

Klíčová slova

Monitorování sítě, zpracování paketů, nástroj pro monitorování, C++, Python, Redis, Xura, Xtend.

Keywords

Network monitoring, packet processing, monitoring tool, C++, Python, Redis, Xura, Xtend.

Citace

KOCMAN, Tomáš. *Nástroj pro monitorování a analýzu aplikačního serveru Xtend*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. František Ščuglík, Ph.D.

Nástroj pro monitorování a analýzu aplikačního serveru Xtend

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Františka Ščuglíka, Ph.D. Další informace mi poskytli členové týmu Xtend. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Kocman
9. května 2017

Poděkování

Rád bych poděkoval vedoucímu své bakalářské práce panu Ing. Františkovi Ščuglíkovi, Ph.D., vedoucímu týmu Xtend Ing. Dalimilovi Hrabovskému a celému týmu Xtend za odborné vedení, rady a čas, které mi při tvorbě této práce poskytli.

Obsah

Kapitola 1

Úvod

Životní cyklus libovolného projektu se skládá z jednotlivých etap, které bychom si mohli představit jako kostky stavebnice, které skládáme na sebe a tím vytvoříme hotovou stavbu. Ta v našem případě představuje cíl projektu, kterého se snažíme dosáhnout a kostky představují jednotlivé etapy vývoje našeho projektu.

První etapou je zjištění požadavků. Mezi ně patří například otázka funkčnosti, designu nebo integrace s ostatními systémy. Druhá etapa obsahuje podrobnou analýzu problému. Vytváří se konceptuální model systému, pomocí kterého se snažíme co nejlépe zachytit skutečnost. Dále odhady nákladů, zdrojů a časový odhad. Ve třetí etapě vytváříme již konkrétní návrh, neboli implementační model. Ten nám má pomoci odhalit všechny technické aspekty projektu ještě před samotnou implementací, abychom se mohli předem vyhnout případným problémům. V následující etapě se projekt implementuje. Po skončení této fáze by měla být implementace již kompletní. Pátá etapa slouží k testování projektu. Je spuštěn zkušební provoz, hledají se chyby, které se ladí. Poslední, šestá etapa, se stará o udržování projektu v provozu.

Tato práce se zaměřuje na pátou etapu, což je testování. Cílem práce je navrhnout, implementovat, otestovat a dodat do produkce systém pro monitorování zpráv. Práci zadala firma Xura, Inc., konkrétně tým Xtend. Monitorování zpráv má probíhat na úrovni aplikačního serveru Xtend, kdy server produkuje informace o jednotlivých zprávách, které monitorovací systém sbírá a následně analyzuje.

Aplikace má být určena převážně zákaznické podpoře a integračním inženýrům, kteří řeší problémy při nasazení Xtend serveru k zákazníkům. Má jim ulehčit vyhledávání příčin problémů, kterých se server Xtend dopustí tím, že přehledně zobrazí podrobné informace o zprávách tak, jak byly postupně zprácovávány serverem.

Práce je rozčleněna do osmi kapitol. Kapitola 2 podrobněji popisuje, jak se paket v systému zpracovává a problematiku zachytávání dat na síti mezi jednotlivými stanicemi. Následující kapitola 3 se věnuje nejznámějším běžně dostupným nástrojům pro zachytávání dat na síti a jejich interpretaci. Na konci kapitoly je objasněno, proč jsou pro naše účely nedostačující. Kapitola 4 objasňuje pojem aplikační server Xtend stejně jako firmu Xura, Inc. s týmem Xtend, pro který je nástroj tvořen. Kapitola 5 a 6 je zaměřena na popis návrhu a implementace monitorovacího nástroje. V posledních dvou kapitolách 7 a 8 je popsán způsob testování aplikace a celkové shrnutí spolu s plánem dalšího vývoje.

Kapitola 2

Monitorování datových přenosů

Oblast minotorování sítí se vyvíjí spolu se sítěmi samotnými. Od vzniku sítí je potřeba komplexního monitorování. To nám pomáhá především při hledání a následném řešení problémů nebo například v oblasti bezpečnosti. Tato kapitola popisuje typ monitorování, kdy je v síti monitorovací stanice, která zachycuje komunikaci na síti a následně ji analyzuje.

Síťová zařízení filtrují přijaté pakety podle MAC adresy a doručují je systému podle daných pravidel:

- Unicastové pakety, které jsou zaslány na jednu z adres síťového adaptéru, tzn. všechny pakety explicitně zaslány konkrétnímu zařízení na síti.
- Multicastové pakety zaslány na multicastovou adresu síťového adaptéru nebo všechny pakety zaslány zařízení bez ohledu na adresu příjemce. Některé síťové adaptéry mohou být nastaveny tak, aby přijímaly pakety pro specifickou adresu, jiné zase pro libovolnou adresu.
- Pakety typu broadcast.

Ovladač síťového adaptéru pošle kopii přenášeného paketu monitorovacímu procesu, takže tento paket bude viditený monitorovacím mechanismem. Konkrétněji bude popsáno zpracování paketů v sekci 2.1.

Pokud se budeme snažit monitorovat pakety jiné než typu unicast daného zařízení nebo multicast či broadcast, adaptér se bude muset přepnout do promiskuitního režimu, aby se výše popsaný filtr vypnul a monitorovací software obdržel kopii všech paketů. Tento režim bude blíže popsán v sekci 2.2.

Jesliže nepracujeme na sdílené ethernetové síti, ale na přepínané ethernetové síti, je potřeba rovněž provést patřičné akce zajišťující to, že síťový provoz, který nás zajímá, bude doručen na síťový adaptér monitorovacího zařízení. Toto není přirozené chování přepínané ethernetové sítě. Možnosti, které se nabízí k řešení, budou popsány v sekci 2.3 [18].

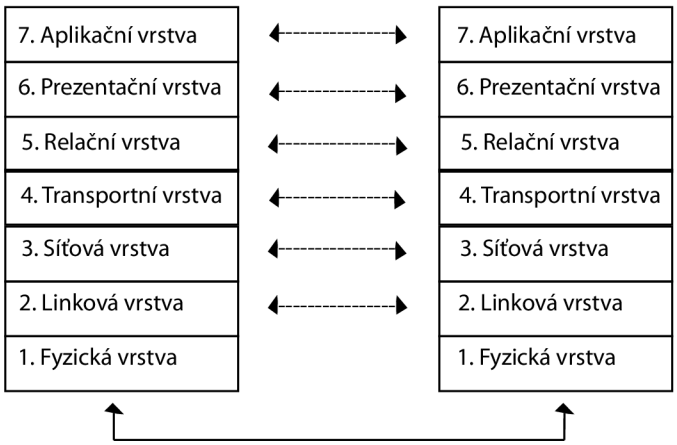
V poslední sekci 2.4 bude vysvětlena problematika zachytávání paketů na bezdrátových sítích typu 802.11 a s tím spojený monitorovací režim [18].

2.1 Zpracování paketů

V této kapitole bude vysvětleno, jak se pakety zpracovávají a s tím spojený sedmivrstvý model ISO/OSI. V první části bude popsán vrstvený model a v další části budou popsány operace s daty, které musí proběhnout na každé z těchto vrstev při odesílání a zpracování paketu [7].

2.1.1 Vrstvený model

Každá vrstva je zodpovědná za svou část komunikace. Vrstvy jsou výhodné, neboť dovolují programátorům rozvíjet jednotlivé části systému nezávisle na sobě, obvykle různými lidmi, kteří mají rozdílné specializace. Původní a nejčastěji zmiňovaný koncept protokolového vrstvení je založený na standardu OSI (Open Systems Interconnection), který je definován mezinárodní organizací pro normalizaci (ISO). Ačkoli je OSI model navržen tak, že má sedm vrstev, aby byla dodržena žádoucí modularita architektury implementace, běžně se zmiňuje TCP/IP architektura, která čítá čtyři vrstvy. Přestože dnes můžeme říct, že TCP/IP model "vyhrál", většinu nápadů a dokonce i celých protokolů TCP/IP model převzal z OSI modelu. Dále bude proto rozebírán OSI model [7].



Obrázek 2.1: ISO/OSI model

Jak již bylo nastíněno, každá vrstva má různou zodpovědnost. První, fyzická vrstva, popisuje metody pro přenos digitálních dat skrze komunikační médium, jako je například optický kabel nebo telefonní linka. Definuje elektrické a fyzikální vlastnosti zařízení. Obsahuje rozložení pinů, napěťové úrovně a specifikuje vlastnosti kabelů.

Na druhé, spojové vrstvě, je definováno mnoho ethernetových a WLAN standardů. Spojová vrstva zahrnuje protokoly a metody pro nastavení konektivity se sousedními uzly sdílejícími totožné médium. Zde se uspořádávají data z fyzické vrstvy do logických celků nazývaných rámce. Některé sítě druhé vrstvy, například DSL spojují pouze dvě zařízení. Pokud se ke sdílené síti může připojit vícero zařízení, mluvíme o multi–access síti. Wi-Fi nebo Ethernet jsou nejznámějšími zástupci sítí druhé vrstvy typu multi–access a používají specifické protokoly pro přístup ke sdílenému médiu v daném čase.

Třetí, síťová vrstva, poskytuje interoperabilní formát paketu (například pro paketové TCP/IP sítě), který může být přenesen různými technologiemi druhé vrstvy. Stará se o směrování a adresování v síti. Poskytuje spojení mezi zařízeními, které spolu přímo nesousedí. Rovněž poskytuje funkce k zajištění přenosu dat různé délky od zdroje k příjemci skrze jednu nebo více propojených sítí při zachování kvality služby, kterou požaduje spojová

vrstva. Obsahuje směrovací funkce a je schopna zasílat zprávy o problémech při doručování dat.

Nad třetí vrstvou jsou teoreticky protokoly implementované pouze koncovými zařízeními, což zahrnuje i čtvrtou, transportní vrstvu. Tato vrstva zajišťuje přenos dat mezi koncovými uzly. Jejím účelem je poskytnout takovou kvalitu služeb, jakou požadují vyšší vrstvy. Vrstva nabízí spojově (TCP) a nespojově (UDP) orientované protokoly (TCP a UDP platí pouze pro architekturu TCP/IP).

- TCP zajišťuje spolehlivý přenos dat. Jedná se o přenosy, kde nesmí být data ztracena, například pro přenos souborů, e–mailů nebo WWW stránek. Jednotkou posílané informace je TCP segment.

- UDP zajišťuje přenos dat bez záruky, že data dojdou nebo budou obdržena správně. Využívají jej aplikace, u kterých požadujeme rychlý přenos a snažíme se vyhnout většímu zdržení paketů. Používá se například pro systém DNS, VoIP nebo streamovaného videa.

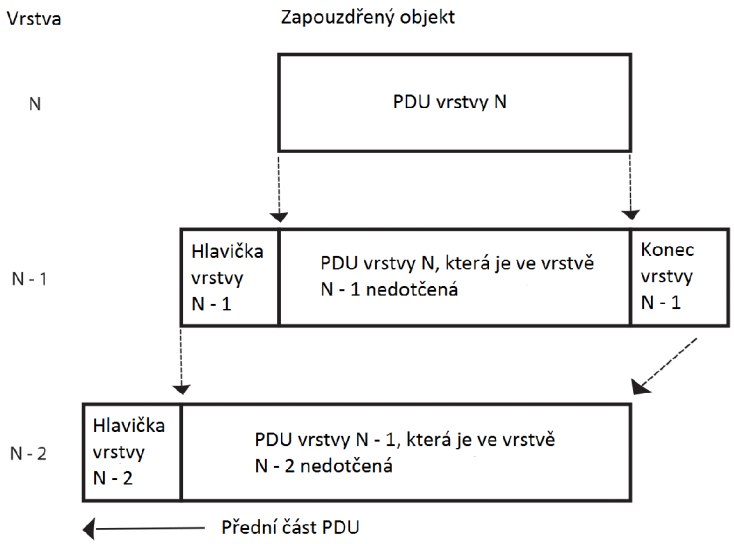
Pátá, relační vrstva, organizuje a synchronizuje dialog mezi spolupracujícími relačními vrstvami obou systémů a řídí výměnu dat mezi nimi. Umožňuje vytvoření a ukončení relačního spojení, synchronizaci a obnovení spojení. Příkladem mohou být soubory cookies, které používá webový prohlížeč během sezení, NetBIOS nebo RPC.

Funkcí šesté, prezentační vrstvy, je transformovat data do tvaru, který používají aplikace použitím šifrování, konvertování nebo komprimace. Formát dat se může lišit na obou komunikujících systémech. Navíc dochází k transformaci pro účel přenosu dat nižšími vrstvami. Mezi funkce patří například převod kódů a abeced, modifikace grafického uspořádání nebo přizpůsobení pořadí bytů. Vrstva se zabývá jen strukturou dat, ne jejich významem.

Účelem poslední, sedmé aplikační vrstvy je poskytnout aplikacím přístup ke komunikačnímu systému a umožnit tak jejich spolupráci. Aplikace obvykle implementují jejich vlastní protokoly a ty jsou nejvíce viditelné uživatelům. Existuje velké množství protokolů sedmé vrstvy a programátoři stále vytváří nové. Zároveň je sedmá vrstva ta, kde probíhá nejvíce inovací a která je nejvíce vyvíjena. Do této vrstvy se řadí například služby DNS, DHCP nebo SMTP [7].

2.1.2 Multiplexing, demultiplexing a zapouzdření

Největším benefitem vrstvové architektury je umožnění protokolového multiplexingu. Tato forma multiplexingu dovoluje koexistenci různých protokolů na totožné infrastruktuře. Multiplexing se může objevit na různých vrstvách a každá z vrstev má identifikátor sloužící pro určení, který protokol nebo stream informací patří dohromady. Například na spojové vrstvě většina technologií (Ethernet, Wi-Fi) obsahují identifikátor v každém paketu, indikující, který protokol je přenášen v rámci druhé vrstvy, například protokol IP. Jestliže objekt, protocol data unit (PDU), například paket nebo segment nějaké vrstvy je přenášen nižší vrstvou, nazýváme to zapouzdřením. Čili různé objekty vrstvy N můžou být multiplexovány dohromady použitím zapouzdření vrstvy $N-1$. Zapouzdření názorně ukazuje obrázek 2.2.



Obrázek 2.2: Zapouzdření dat

V obrázku lze vidět, že každá vrstva má svoji vlastní PDU. Pokud vrstva poskytuje PDU vyšší vrstvy, obvykle se zavazuje k tomu, že nebude s daty vyšší vrstvy manipulovat, což je podstata zapouzdření. Ve většině případů vrstva vloží před PDU svou charakteristickou hlavičku a v některých případech i trailer, což jsou data přiložena za konec PDU. Hlavička je použita pro multiplexing dat při odesílání a demultiplexing na základě identifikátoru na straně příjemce. V TCP/IP sítích tento identifikátor může být například MAC adresa, IP adresa nebo číslo portu [7].

2.2 Promiskuitní režim

V ethernetových sítích může být síťové rozhraní přepnuto do promiskuitního režimu, což zařízení umožní obdržet všechnu komunikaci, která se objeví na síťovém rozhraní, dokonce i tu, která není explicitně zaslána tomuto zařízení. V počátcích ethernetu, kdy médiem byl sdílený kabel, dovoloval tento režim "odposlouchávat" komunikaci ostatních zařízení velmi lehce. Tento režim je potřeba například u aplikací, které měří výkonnost nebo hledají bezpečnostní slabiny na lokálním síťovém segmentu. Smysl má rovněž pro zákonné odposlouchávání. Vyšší protokoly obsahovaly citlivé informace jako například uživatelská hesla, takže si každý mohl tyto informace jednoduše přečíst v ASCII kódování. Dva důležité faktory tuto skutečnost ovšem výrazně ztížily. Vývoj síťového zařízení přepínače a šifrování dat protokolů vyšších vrstev. Přepínané ethernetové sítě budou rozebrány v sekci 2.3 [8].

Pokud chceme z druhé vrstvy obdržet všechny pakety, v linuxových systémech jsou pro to dvě metody. Původní, která je více rozšířená, avšak méně flexibilní je vytvořit socket typu *SOCKET_PACKET*. Novější metodou, která přináší více filtrovacích a výkonnostních vlastností je vytvořit socket rodiny *PF_PACKET*. K tomu musíme mít požadovaná uživatelská práva, podobná jako při vytváření *raw* socketu. Druhý parametr zvolíme *SOCK_RAW*, pro

získání kompletních paketů druhé vrstvy. Třetím parametrem musí být nenulová hodnota specifikující typ ethernetového rámce.

Kompletní nastavení socketu, pokud chceme obdržet všechny pakety, může být například jedno z těchto dvou:

- *fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));*
- *fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP));*

První možnost zajistí, že socket bude schopen obdržet rámce všech protokolů, které se objeví na spojové vrstvě. Druhá možnost umožní obdržet pouze IPv4 rámce [9].

2.3 Sítě typu Ethernet

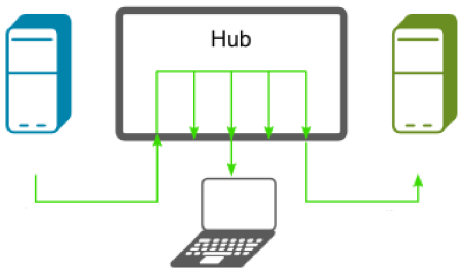
Pokud se snažíme zachytit síťová data mezi stanicí, na které běží monitorovací software a ostatními stanicemi, lze tato data zachytit na síťových rozhraních monitorovací stanice, přes která jsou pakety odesílána nebo přijímána. Žádné další speciální nastavení není potřeba. Pokud chceme zachytit data mezi dvěma procesy, které se nacházejí na stejné stanici, je potřeba nastavit monitorování na tzv. loopback rozhraní.

Loopback rozhraní je virtuální síťové rozhraní TCP/IP internetového protokolu. Slouží k tomu, aby spolu mohly komunikovat procesy na totožném zařízení.

Jestliže se pokoušíme zachytit síťová data, která nepatří monitorovací stanici, tzn. data, která nejsou odeslána ani přijímána monitorovací stanicí, ale jinými dvěma nebo vícero komunikujícími stanicemi na ethernetovém segmentu, je potřeba zachytávat v promiskuitním režimu. Pokud navíc monitorujeme na přepínané ethernetové síti, musí být stanice speciálně nastavena, což bude popsáno v následujících sekcích [18].

2.3.1 Sdílená ethernetová síť

V počátcích internetových sítí existovaly ethernetové sítě pouze sdílené. Používaly hub, což je sdílené médium, které propojuje jednotlivé ethernetové uzly dohromady. Když někdo pošle paket, hub ho vyšle všemi síťovými rozhraními, takže daný paket obdrží všechny zařízení na síti. Pokud je v tomto případě stanice v promiskuitním režimu, zachytí všechny pakety jdoucí skrze hub, tedy komunikaci všech zařízení v síti. Dnes jsou sdílené sítě populární obzvláště díky WLAN technologiím, což je blíže popsáno v sekci 2.4.

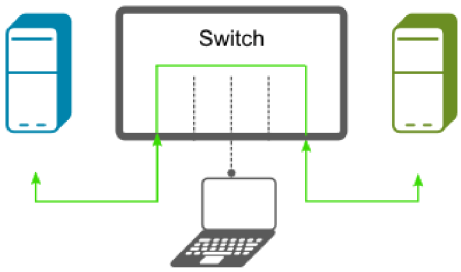


Obrázek 2.3: Segment se zařízením typu hub

2.3.2 Přepínaná ethernetová síť

Dnes jsou typické ethernetové sítě, které využívají přepínač, spojující všechny uzly sítě dohromady. Tento přístup vysoce zvyšuje výkonnost sítě, na druhou stranu znesnadňuje zachytávání paketů.

Přepínač si interně udržuje informace o tom, jaké zařízení je připojeno k jakému síťovému rozhraní. Podle těchto pravidel rozhoduje, kam bude posílat dané pakety, konkrétně unicastové, multicastové a broadcastové pakety. Tato pravidla se řídí podle MAC adres stanic připojených k přepínači.



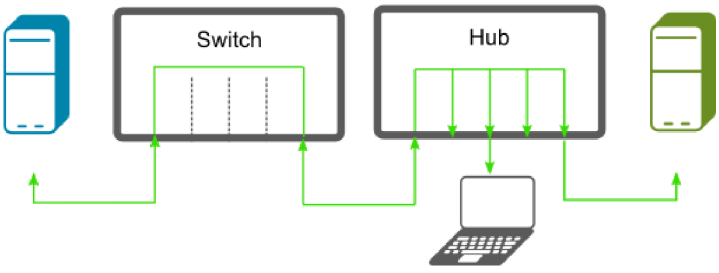
Obrázek 2.4: Segment se zařízením typu přepínač

Unicastové pakety, které nejsou zaslány monitorovacímu zařízení tedy přepínač nepošle přes síťové rozhraní, na kterém je toto zařízení připojeno. Síťový adaptér monitorovacího zařízení obdrží unicastové pakety zaslané přímo jemu, žádné jiné. Promiskuitní režim v tomto případě nemá význam, jelikož pakety k síťovému adaptéru nedorazí.

V následujících kapitolách budou popsány možnosti, jak tento problém řešit.

2.3.3 Použití ethernetového hubu

Jestliže máme k dispozici hub, umístíme ho mezi zařízení, která chceme minotorovat. Například mezi přepínač a zařízení, která chceme monitorovat nebo mezi dva přepínače.



Obrázek 2.5: Použití hubu

Některé huby (přepínané huby) se chovají jako přepínače, tzn. přepínají pakety podle MAC adres. V tomto případě ztrácí hub zcela svůj význam, neboť monitorovacímu zařízení nezašle pakety, které nepatří přímo tomuto zařízení.

Nevýhodou je, že tato metoda degraduje výkon sítě a není vhodná pro řešení problémů na síti. Navíc potřebujeme fyzický přístup k zařízením sítě, což často není možné. Dalším úskalím je získat samotný hub, neboť se dnes v praxi nevyužívá.

2.3.4 Monitorovací režim přepínače

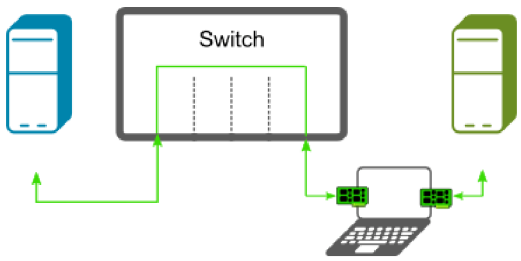
Některé přepínače disponují tzv. monitorovacím režimem. Tyto přepínače jsou schopné své síťové rozhraní, ke kterému je připojena monitorovací stanice, přepnout do monitorovacího režimu. Procedury konfigurace závisí na modelu přepínače. Některé podporují pouze režim příkazové řádky, jiné specializovaný SNMP klientský software nebo i webové rozhraní.

Na takto nakonfigurované síťové rozhraní bude přepínač zasílat kopii paketů, které obdrží na ostatních rozhraních, takže monitorovací stanice obdrží všechny pakety z daného síťového segmentu. Ne všechny přepínače ovšem podporují monitorování jakékoli komunikace jdoucí přes tento přepínač. Některé dovolují monitorování komunikace například jen některých aplikačních portů. V takovém případě monitorovací stanice neobdrží veškerou komunikaci, ale pouze přepínačem nakonfigurovanou.

Nevýhodou těchto přepínačů je cena. Ačkoli je cena vysoká, dnes je přístupná široké veřejnosti, v minulosti to tak ovšem nebylo a často se muselo přistoupit k jinému řešení. Rychlost síťového rozhraní monitorovací stanice musí být stejná nebo vyšší než rychlost monitorovacího rozhraní na přepínači, jinak hrozí ztráta paketů.

2.3.5 Metoda machine-in-the-middle

Jde o zapojení speciálně vybaveného zařízení mezi přepínačem a dalším uzlem v síti. Toto zařízení disponuje dvěma fyzickými síťovými rozhraními, které spolu tvoří most. Pakety se přeposílají z jednoho rozhraní na druhé, takže pokud monitorovací stanice monitoruje na jednom z těchto rozhraní, obdrží kopii každého paketu.



Obrázek 2.6: Ukázka zapojení machine-in-the-middle

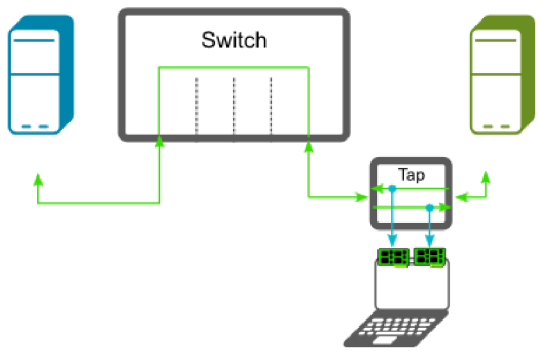
Na linuxových systémech lze most nakonfigurovat například pomocí programu *brctl*, na systémech rodiny BSD lze použít program *brconfig*, obdobně lze použít software pro vytváření mostu na systémech Windows.

Nevýhoda této metody je nutnost speciální konfigurace monitorovacího zařízení.

2.3.6 Síťové zařízení tap

Mnoho výrobců nabízí tzv. síťový tap, což je zařízení, které umožňuje monitorování sítě podobně jako metoda machine-in-the-middle. Tap musí mít minimálně tři rozhraní – vstupní

a výstupní síťové rozhraní a monitorovací rozhraní. Tap přeposílá pakety mezi rozhraními a kopii zasílá monitorovacímu mechanismu. Většina modelů má monitorovací rozhraní dvě, kdy každé je zapojeno na síťové rozhraní monitorovací stanice a zasílá kopie paketů daného směru. Pokud monitorovací software nepodporuje monitorování dvou síťových rozhraní současně, je potřeba spustit dvě instance tohoto softwaru a jejich výstupy sloučit [13].



Obrázek 2.7: Zapojení zařízení tap

Síťový tap nachází uplatnění kromě monitorování paketů například v network intrusion detection systems, VoIP nahrávání, síťových sondách, RMON sondách a jiných monitorovacích systémech.

2.4 Bezdrátové sítě

Tato sekce se bude zabývat bezdrátovými sítěmi standardu IEEE 802.11. Nejprve je potřeba si uvědomit, který z těchto případů nás zajímá:

- Pokud se snažíme zachytit komunikaci mezi zařízením, na kterém běží monitorovací software a jinými zařízeními v síti a zajímají nás pouze regulérní síťová data, jsme schopni toho docílit jednoduše monitorováním daného síťového rozhraní, přes které jsou data přijímána a odesílána.
- Jestliže chceme zachytit komunikaci, která nepochází nebo není zaslána monitorovací stanici podobně jako na ethernetovém segmentu, popřípadě pokud se snažíme zachytit kontrolní pakety 802.11 nebo informace o paketech na rádiové vrstvě, je potřeba zachytávat v tzv. monitorovacím režimu. Ten bude blíže popsán v následující sekci.

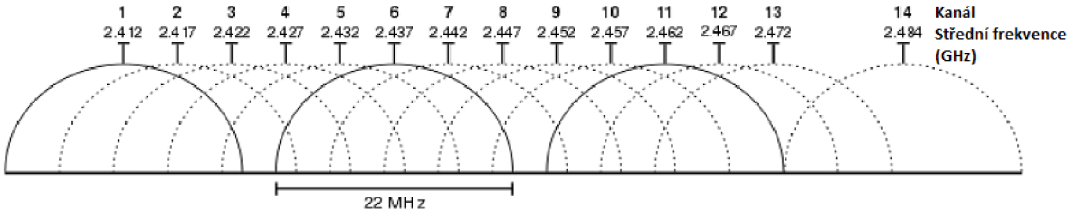
Pokud monitorujeme komunikaci na bezdrátové síti bez jakékoli další konfigurace, zachycené pakety budou mít tzv. falešné ethernetové hlavičky. V tomto případě neuvidíme kontrolní nebo 802.11 řídící pakety. Hlavičky 802.11 paketů budou síťovým ovladačem přeloženy na falešné ethernetové hlavičky.

Promiskuitní režim je v praxi nakonfigurovatelný na většině 802.11 adapterů, ale málokdy skutečně funguje narozdíl od promiskuitního režimu ethernetových adaptérů. Samotný pokus o přepnutí do promiskuitního režimu může selhat. Potom bude adaptér zachytávat pakety pouze odeslané nebo přijaté vlastní stanicí.

V bezdrátových sítích jsme schopni zachytit tyto typy paketů:

- Datové pakety, které jsou obvykle doručovány monitorovacímu mechanismu s falešnými ethernetovými hlavičkami syntetizovanými z 802.11 hlavičky. Skutečnou 802.11 hlavičku u tohoto typu paketu neuvidíme.
- Nedatové pakety, které můžeme zachytit v monitorovacím režimu. Takové pakety budou zachyceny s 802.11 hlavičkou.
- Řídící pakety, které slouží k řízení uzlů bezdrátových sítí, které ovšem nad druhou vrstvou ISO/OSI modelu ztrácí význam. Většina ovladačů síťových karet tyto pakety zahazuje, tudíž vůbec nepřijdou k monitorovacímu mechanismu. Pokud ovšem tyto pakety ovladač podporuje, lze je v monitorovacím režimu zachytit.
- Nízkoúrovňové kontrolní pakety sloužící k synchronizaci přístupových kanálů mezi uživateli na síti. Stejně jako řídící pakety, jsou důležité pouze na druhé vrstvě ISO/OSI modelu. Většina adaptérů je rovněž zahazuje, popřípadě můžou být zachyceny v monitorovacím režimu jako v předchozím případě.

802.11 používá rádiové frekvence v rozsahu 2412–2484 MHz. V některých zemích ovšem nejsou povoleny některé z těchto frekvencí. Tato frekvence je rozdělena do 14 kanálů, očíslovaných 1–14.



Obrázek 2.8: Frekvence kanálů

Například v Evropě jsou využity kanály 1–13, v Japonsku 1–14 a ve Spojených Státech 1–11. Frekvenční rozsahy kanálů se částečně překrývají, tudíž nejsou zcela nezávislé. Kanály 1, 6 a 11 se nepřekrývají, jsou proto neoficiálně standardem nezávislosti bezdrátových kanálů. Uživatel si musí na síťovém adaptéru nebo přístupovém bodu zvolit kanál, který chce pro komunikaci používat. Data potom budou obdržena a zasílána pouze tímto kanálem. Tento typ filtrování nemůže být zakázán, avšak některé speciální síťové adaptéry jsou schopny zachytit data z vícero kanálů zároveň [18].

2.4.1 Monitorovací režim

Síťovému adaptéru nebo přístupovému bodu za běžných okolností uživatel zvolí SSID (Service Set Identifier), což je název sítě. Pokud má více přístupových bodů stejné SSID, potom je to nazýváno ESSID (Extended SSID). Adaptér dále filtruje komunikaci na základě SSID a nezahazuje pouze ty pakety, které jsou stejného SSID.

V monitorovacím režimu je filtr, který zahazuje pakety na základě SSID vypnutý a všechny pakety libovolných SSID jsou zachyceny. Za předpokladu, že zachytáváme v promiskuitním režimu, 802.11 adaptér doručí systému pouze pakety shodující se v SSID, ostatní

budou zahozeny. Ačkoli adaptér může tyto pakety obdržet na rádiové úrovni, pokud se neshoduje SSID, zahazuje je.

Pokud tedy chceme zachytit všechna data, která můžou být obdržena, musíme daný adaptér přepnout do monitorovacího režimu, někdy nazývaným rfmon režimem. Potom ovladač nastaví síťový adaptér tak, aby doručoval systému všechny pakety bez ohledu na SSID. Dále záleží na ovladači, zdali adaptér odloučí od jeho SSID. Pokud by byl odloučen, stanice nebude schopna adaptér použít pro komunikaci s jinými uzly sítě. V opačném případě může být adaptér stále použit pro komunikaci s okolím. V případě, že adaptér bude odloučen od SSID a zařízení nemá žádný další síťový adaptér, nebude schopen například přeložit doménová jména nebo uložit zachycenou komunikaci na síťový souborový systém.

Kapitola 3

Nástroje pro monitorování a analýzu datových přenosů

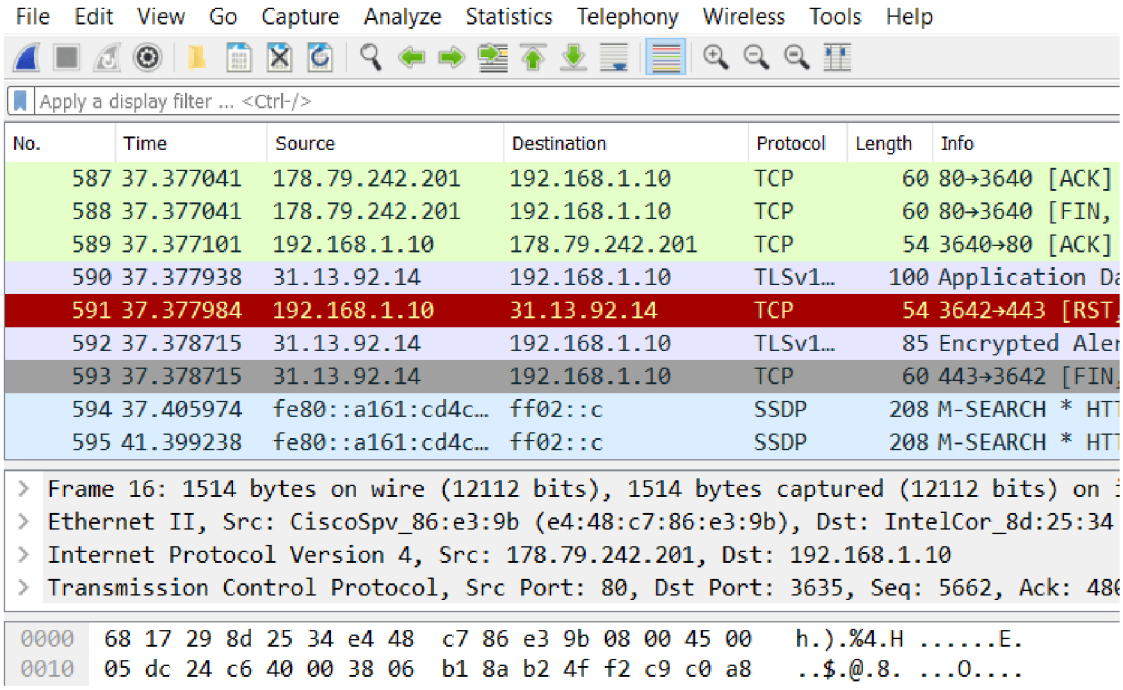
V síti se může denně vyskytnout mnoho různých problémů, počínaje infikací koncových zařízení škodlivým softwarem, až po složitý konfigurační problém na straně směrovačů. Je nemožné všechny problémy vyřešit okamžitě, ale je vhodné být připraven jak znalostmi, tak patřičnými nástroji, v našem případě analyzátorem paketů. Většina problémů pramení z paketové úrovně, kde může například hezky vypadající aplikace skrývat velmi špatnou implementaci komunikačního protokolu a zahlcovat celý síťový segment škodlivými daty. Musíme se tedy s analyzátorem paketů ponořit na úroveň těchto paketů a zjistit, kde nastal problém. Analýza paketů je proces zachytávání a interpretování dat, která zrovna projdou přes monitorovaná síťová rozhraní za účelem pochopení toho, co se aktuálně na síti děje. V této kapitole budou představeny dva z nejpoužívanějších analyzátorů paketů, Wireshark v sekci 3.1 a Tcpdump v sekci 3.2 [6][17].

3.1 Wireshark

Wireshark čítá mnoho benefitů, které zde budou postupně popsány a díky kterým se ho vyplatí použít na jakýkoli typ analýzy sítě. Wireshark necílí jen na síťové experty, ale i na začátečníky, kteří nemají s analýzou paketů zkušenosti. Exceluje především v počtu implementovaných protokolů. To sahá od známých protokolů, například IP, DHCP až po ty pokročilejší a dokonce proprietární protokoly jako AppleTalk nebo BitTorrent. Protože je Wireshark vyvíjen pod open source licencí, podpora nových protokolů přichází s každou další aktualizací. V případě, že nepodporuje požadovaný protokol, je zde možnost vlastní implementace modulu, který tento protokol bude podporovat a zaslat jej vývojářům Wiresharku pro oficiální zahrnutí [6].

Rozhraní k ovládání aplikace je velmi jednoduché k použití. Uživatel má na výběr mezi grafickou a konzolovou verzí. Konzolová se nazývá TShark a manipulace s ní probíhá plně přes terminálové okno. Dále se ovšem budeme zaobírat verzí s grafickým rozhraním. Ta je velmi jednoduchá a přímočará k ovládání. Nabízí mnoho vlastností, které usnadní použití jako například zbarvení určitých částí na základě protokolu a detailní grafickou reprezentaci dat. Narozdíl od složitějších terminálových analyzátorů jako Tcpdump nabízí Wireshark grafické rozhraní, které usnadní začátečníkům orientaci v různých protokolech. Wireshark je vydáván pod GPL licencí, je tudíž zdarma pro osobní i komerční užití.

Když se používá volně distribuovaný software, musí se počítat s tím, že může být i bez oficiální podpory. Proto open source komunita spoléhá na samotné uživatele aplikace k poskytnutí základní podpory. Wireshark má ovšem velkou a velmi aktivní komunitu uživatelů i vývojářů, kteří tento projekt udržují. Na webu lze najít odkazy na různé druhy podpory jako online dokumentace, Wikipedie zahrnující vývoj a podporu nebo e-mailové konference, kde přispívají hlavní vývojáři. Wireshark podporuje všechny moderní operační systémy, což jsou platformy založené na Linuxu, dále Mac OS X a Windows.



Obrázek 3.1: Okno Wiresharku

Většinu času uživatel stráví v hlavním okně aplikace, kde jsou zobrazeny všechny zachycené pakety. Informace o paketech jsou formátovány a dále rozděleny do více sekcí. V hlavním okně se nacházejí tři panely, kde jsou zobrazeny informace o zachycených paketech. Po zvolení jednoho z paketů vidíme v každém panelu jiné informace. První, největší panel, obsahuje seznam všech paketů spolu se základními informacemi, mezi které patří číselné označení paketu, čas zachycení, zdrojová a cílová adresa, použitý protokol, délka paketu v bytech a obecná informace o paketu. Prostřední panel ukazuje detail paketu, hierarchii podrobných informací. Po rozkliknutí libovolné vrstvy v tomto panelu se rozbalí podrobnosti paketu z dané vrstvy. Ve spodním panelu jsou zobrazena čistá nezpracovaná data, což pro někoho může být matoucí. Data jsou reprezentována v původním formátu, jak přišla na síťové rozhraní.

Wireshark umožňuje použití filtrů, které velmi urychlují hledání požadovaných paketů. Filtr je jednoduchý výraz ve formě predikátové logiky, který definuje kritéria pro inkluzi a exkluzi paketů. Pokud jsou zachyceny pakety, které pro potřeby uživatele nejsou důležité, mohou být jednoduše pomocí filtru odstraněny z výpisu. Filtry mohou vypadat následovně:

- dns || dhcp || http

- eth.dst == 03:41:44:12:36:10 && tcp.port == 443

První filtr specifikuje zobrazení pouze těch paketů, které mají jako protokol sedmé vrstvy jeden z protokolů DNS, DHCP nebo HTTP. Druhý filtr vyjadřuje pravidlo, kde musí být cílová adresa druhé vrstvy rovna *03:41:44:12:36:10* a zároveň zdrojový nebo cílový port TCP musí být roven *443*.

3.2 Tcpdump

Tcpdump je jeden z prvních open source nástrojů pro analýzu paketů. Byl vytvořen roku 1987 lidmi z Lawrence Berkeley Laboratory. Jde o konzolový nástroj, který je spustitelný na většině linuxových systémů, Solaris, AIX, Mac OS X, BSD a HP UX. Existuje i WinDump, což je verze pro systémy Windows.

Stejně jako Wireshark, i Tcpdump využívá pro svou činnost knihovnu libpcap. Ta obsahuje soubor funkcí nezávislých na platformě pro sběr paketů ze síťových rozhraních a pro analýzu těchto paketů. Tcpdump poskytuje rozhraní pro komunikaci s libpcap, která dále komunikuje s ovladačem síťového rozhraní, který ovládá dané zařízení. Tcpdump, WinDump i Wireshark využívají pro svou činnost Berkeley Packet Filter, což poskytuje přístup ke spojové vrstvě, kde může být pomocí filtru specifikováno, která data chce aplikace obdržet a která ne [17].

Tcpdump může být použit pro většinu mechanismů, které monitorují síťový provoz v promiskuitním režimu. Je ovšem potřeba myslet na několik limitací:

- Tcpdump je terminálový nástroj. Uživatel musí mít dobrou znalost všech přepínačů, které specifikují zobrazení informací o paketech, aby mohl efektivně pracovat. Nenabízí tedy jednoduché uživatelské rozhraní, které by mohl začátečník použít.
- Jak již bylo vysvětleno, pokud paket zablokoval firewall, router nebo se vyskytujeme na přepínané síti, monitorovací stanice daný paket nemůže zachytit. Tcpdump, stejně jako ostatní monitorovací aplikace, zachytí pouze tu komunikaci, která je směrována na stanici, kterou monitoruje.
- Pokud požadujeme znovu "přehrát" komunikaci tak, jak byla zachycována nebo chceme podrobnější analýzu, musíme použít další nástroje, například tcpreplay nebo tcpopera.

Když spustíme Tcpdump s přepínačem *-c*, bude zachytávat pakety, dokud nebude proces ukončen signálem SIGINT, signálem SIGTERM nebo se nezachytí specifikovaný počet paketů. Bez tohoto přepínače se bude zachytávat, dokud nebude proces ukončen signálem SIGINT nebo SIGTERM.

V okamžiku, kdy je zachytávání ukončeno, Tcpdump reportuje o následujících informacích:

- Počet zachycených a zároveň zpracovaných paketů.
- Počet paketů obdržených filtrem. Tento výstup záleží na operačním systému a na tom, jak je nakonfigurován. Pokud je filtr specifikován na příkazové řádce, na některých systémech může být výstupem počet paketů bez ohledu na to, zdali byly pakety propuštěny tímto filtrem. A pokud filtrem propuštěny byly, nemusel je Tcpdump vůbec přečíst a zpracovat. Na jiných systémech to může být počet paketů zachycených filtrem a zároveň zpracovaných Tcpdumpem.

- Počet paketů zahozených jádrem systému. Jádro paket zahazuje například z důvodu nedostatku paměti pro monitorovací mechanismus v systému, kde je spuštěn Tcpdump.

Příkladové zadání filtrů mohou být například:

tcpdump ip and not net localnet
Zachytávání paketů, které nepochází, ani nejsou směrovány na localhost.

tcpdump -i eth0
Naslouchání na rozhraní eth0.

tcpdump -n dst host 192.168.1.1
Zachytávání paketů, které jsou zaslány na adresu 192.168.1.1. Zobrazení IP adresy a čísla portů.

3.3 Zhodnocení dostupných nástrojů

Všechny zmíněné nástroje fungují na podobném principu. Sbírají síťovou komunikaci tak, že naslouchají na síťovém rozhraní a každý paket, který tímto rozhraním projde zkopírují a následně analyzují. Jsou schopné zobrazovat zachycené pakety v reálném čase, vytvořit podrobnou analýzu o paketech a podporují použití filtrů pro bližší specifikaci požadovaných dat. Uživatelská rozhraní podporují buď grafická nebo rozhraní příkazové řádky, takže jsou tyto nástroje otevřeny širokému spektru uživatelů od začátečníků až po experty.

Požadavek na nástroj pro monitorování aplikačního serveru Xtend vznikl hlavně z interních důvodů. Xtend produkuje soubory pro další analýzu a zpracování, které by se neměly dostat mimo firmu. Ačkoli lze předpokládat, že by šel dopsat modul například do Wiresharku, aby tomuto proprietárnímu formátu porozuměl, pořád jsou zde další limitace. Největší z nich je samotná cloud infrastruktura, která v případě serveru Xtend nedovoluje provádět zásadní změny na síťových rozhraních a dále využívat zdroje tohoto aplikačního serveru.

V kapitole 4 bude stručně popsán Xtend se dvěmi, v dnešní době nejpopulárnějšími službami a jeho vnitřní struktura.

Kapitola 4

Aplikační server Xtend

V této kapitole bude popsána společnost Xura, Inc., její oblast zájmu a jeden z jejích nejúspěšnějších produktů, aplikační server Xtend. Nejprve bude vysvětlena činnost samotné společnosti Xura, poté stručně produkty, které Xtend implementuje, následně podrobněji tři z nejpopulárnějších služeb tohoto produktu, které se v zahraničí aktivně používají a nakonec vnitřní strukturu a logiku fungování.

4.1 Xura

Xura, Inc. je společnost zabývající se digitální komunikací přes mobilní zařízení. Dodává své produkty a pomáhá monetizovat digitální služby největším světovým mobilním operátorům v zahraničí. V říjnu roku 2015 firma Comverse koupila firmu Acision, tímto spojením vznikla Xura.

Vytváří služby spojené s SMS (*Short Message Service*), messagingem a monetizací. Její produkty spadají do dvou kategorií. První je digitální komunikace. Do toho spadá celá infrastruktura mobilní sítě, což obsahuje SMS, MMS, RCS a MaaP, potom zabezpečení proti spamu, řešení SS7 signalizace nebo i Voicemail. Druhá kategorie, monetizace, obsahuje služby *Low Credit* a pokročilé *Value added Services*.

4.2 Xtend

Jak již bylo zmíněno, tato bakalářská práce se zabývá monitorováním aplikačního serveru Xtend, konkrétně monitorováním jeho správné funkčnosti. Xtend je vyvíjen brněnskou pobočkou jedním z týmů Xury. Implementuje služby nad SMS zprávami, které se nasazují u telefonních operátorů. Xtend ke své činnosti využívá dalších produktů Xury, jako například OAM (operation, administration, management), MCO (Message Controller) a AAGP (Acision Application Gateway Provisioning). Xtend je vlastnictvím Xury, takže zde budou popsány nejpopulárnější SMS služby pouze vysokoúrovňově a nebude se zacházet do detailních procesů, které ve službách interně probíhají.

Podle analýz různých států má 50 až 95 % mobilních uživatelů předplacené karty. Tito uživatelé většinou spotřebují svůj kredit dříve, než jsou schopni zaplatit si další. Doba, po kterou má zákazník nulový kredit je právě ta nejdůležitější, protože je zde obrovský potenciál milionů zpráv, které nemůžou být doručeny. Přestože mobilní operátoři nabízejí mechanismy jak překlenout tuto dobu kdy je služba nedostupná, pro uživatele to většinou znamená koupit si credit vouchery a dobít si je. Často se stává, že pro uživatele není tato

možnost v danou chvíli dostupná. V tuto chvíli nastupují na řadu speciální služby, které dokážou překlenout toto období a dynamicky vyřešit tento problém podle aktuální situace uživatele, jako například Collect SMS, Prepaid SMS, Gift SMS nebo Loan SMS.

V sekci 4.2.1 bude popsána služba Collect SMS, v sekci 4.2.2 Prepaid SMS Reply a v sekci 4.2.3 služba Group Messaging. Poté v sekci 4.3 bude vysvětleno, jakým způsobem pracuje aplikační server a jak se konfigurují jednotlivé služby, které si zákazník vyžádá.

4.2.1 Collect SMS

Jak už bylo řečeno, v globálním měřítku používá zhruba 75 % mobilních uživatelů nějakou formu předplaceného používání SMS. Tito uživatelé spotřebují všechny SMS dřív, než si obvykle na konci měsíce dobijí další. Služba Collect SMS se snaží tento potenciální ztracený zisk minimalizovat. Uživatelé, kteří trpí nedostatkem kreditu můžou zaslat SMS s tím, že ji zaplatí druhá strana. Tato SMS zpráva může být účtována individuálně vzhledem k aktuálnímu uživateli. Collect SMS je vhodná převážně pro uživatele, kteří musí nutně zaslat SMS zprávu, ale nemají zrovna dostatek prostředků pro její zaplacení. Příkladem může být:

- Uživatel bez kreditu, který potřebuje napsat kamarádovi.
- Zasílání pracovních zpráv z pracovních zařízení.
- Teenageři snažící se kontaktovat rodiče.
- Děti, které potřebují například vyzvednout ze školy.

Xura Collect SMS tedy nabízí možnost získat zisk navíc tím, že uskuteční platbu, která by za normálních okolností byla odmítnuta. Tuto službu iniciuje odesílatel tím, že před telefonní číslo příjemce vloží předdefinovaný prefix nebo může být iniciována také ze strany sítě, pokud už odesílateli nezbývá žádný kredit. Poté Collect SMS zašle zprávu příjemci se žádostí o přijetí zprávy a účtování této zprávy. Tato zpráva přijde ze *short code*, který je operátorem definovaný. Příjemce odpoví "YES" zpátky na číslo Collect SMS služby a ta přepošle původní zprávu, kterou odesílatel zaslal a účtuje ji příjemci. Tato zpráva má původní číslo odesílatele, takže uživatelé vidí na svých zařízeních kontinuální konverzaci.

Iniciace odesílatelem

Služba je iniciována odesílatelem (A), který před telefonní číslo (MSISDN) příjemce (B) vloží prefix, například 555164538977 (kde *555* je prefix). Prefix je použit, aby zpráva nebyla účtována mobilním operátorem, když dojde na SCP/SMSC (*Service Control Point/Short Message Service Center*) a aby byla správně nasměrována na Xtend server, který dále bude ovládat logiku této zprávy. Xtend obdrží danou zprávu a na základě prefixu rozpozná o jakou službu se jedná. Následně zašle příjemci zprávu o akceptování účtování. B zašle službě odpověď "YES" a služba zašle zprávu od A k B. A může dostat notifikaci o doručení zprávy, to ovšem závisí na konfiguraci u mobilního operátora.

Iniciace sítí

V tomto případě se strana A nemusí zajímat o svůj stav kreditu nebo se pokoušet zaslat zprávu bez dostatku kreditu. Služba Collect SMS je aktivována automaticky z Xura

SMSC, které detekovalo pokus o zaslání SMS s nedostatkem kreditu. Následně je aktivována služba a nabídne odesílateli možnost zaslat SMS pomocí Collect SMS. Když odesílatel odpoví "YES", služba zašle straně *B* zprávu o akceptování zprávy a následuje scénář z iniciace odesílatelem. V tomto druhém případě není potřeba žádného speciálního prefixu pro aktivování služby. Tato možnost je více atraktivní a účinná.

4.2.2 Prepaid SMS Reply

Použití SMS stále splňuje potřeby lidí komunikovat, ale komunikace často stagnuje z důvodu nedostatku kreditu jedné ze stran. Je mnoho situací, kdy si uživatel přeje komunikovat a je zároveň ochotný zaplatit za odpověď. Prepaid SMS má využití převážně v těchto situacích:

- Manažeři, kteří potřebují komunikovat se zaměstnanci, kteří mohou používat své osobní telefony.
- Rodiče, kteří chtějí komunikovat se svými dětmi, které mají omezený kredit.
- Společnosti, které mají nějaký typ závazku se svými zákazníky.
- Kamarádi, kteří si spolu přejí komunikovat.

Pro mobilní operátory nabízí tato služba další profit v situacích, kdy uživatel nemá kredit a přeje si nadále komunikovat. Ve světě není zodpovězeno na miliony zpráv právě z tohoto důvodu, Xura proto nabízí tuto službu, která se snaží minimalizovat počet nedoručených zpráv. Prepaid SMS umožňuje uživateli začít konverzaci tím, že zaplatí druhé straně za možnost odpovědět. Odpověď je validní pouze tehdy, pokud je směrována zpátky k původnímu odesílateli. Zároveň mobilní operátor nemusí investovat jak velké peněžní částky, tak mnoho času do jiných řešení.

Uživateli nabízí Prepaid SMS mnoho benefitů. Služba je *always on*, je uživatelsky přívětivá a nabízí velmi jednoduché ovládání. Není zde potřeba zakládat zvláštní účet, aktivaci nebo modifikaci stávajícího účtu a konfigurace. Služba může být použita uživateli, kteří mají předplacenou kartu nebo si platí tarif.

Chování služby

Strana *A* dá "dárek" straně *B* ve formě Prepaid SMS Reply tím, že přidá prefix služby před telefonní číslo strany *B*. Tento prefix je rozpoznán v síti a je nasměrován na server Xtend, který začne vykonávat službu. Nejdříve se musí zjistit, jestli strana *A* má dostatek kreditu na zaplacení. V kladném scénáři potom služba zašle zprávu straně *B* spolu s informační zprávou. Ta může být například "Info: your reply is prepaid by sender." nebo "Gift SMS. Reply for free." Mobilní operátor může tuto hlášku nakonfigurovat podle svých potřeb. Strana *B* může odpovědět odesílateli tak, jak je zvyklý, bez jakýchkoli změn konfigurace telefonu nebo použití nějakých klíčových slov v textu zprávy.

4.2.3 Group Messaging

Tato služba umožňuje interaktivní používání SMS ve skupině lidí. Dovoluje zasílat zprávy členům ostatních skupin a jednoduše jim odpovídat na jejich zprávy. Funkcionalita je v podstatě totožná se skupinovým chatem běžných sociálních sítí (např. Facebook). Skupiny lze snadno založit, a zasílat mezi nimi zprávy. Zakladatel nebo jiný člen skupiny může pozvat

další členy a jakmile nabídku přijmou, budou jim zasílány všechny zprávy od ostatních uživatelů a je jim rovněž umožněno odpovídat. Hlavním benefitem pro uživatele je možnost vyměňovat si zprávy mezi dvěma a více lidmi ve skupině. Ve skupině mají uživatelé různé role. Například zakladatel skupiny je jejím administrátorem a má jako jediný právo vyloučit členy nebo může nastavovat další parametry skupiny. Po technické stránce se to liší od běžné SMS zprávy rozeslané individuálně všem lidem tím, že při použití této služby přijde na SMSC od odesílatele pouze jedna zpráva, která se nakopíruje a rozešle uživatelům. Využití tato služba nachází převážně:

- V plánování byznys meetingů.
- Při připravě rodinných akcí.
- Při žádosti o pomoc v nouzi.
- V povídání si s přáteli.

Služba obsahuje mnoho *Group Messaging Service Numbers*, aby jednotlivé skupiny mohly být jednoznačně identifikovány. Založení a používání této služby se skládá ze sedmi kroků:

- Uživatel A chce vytvořit skupinu s názvem WorkingGroup. Pošle SMS zprávu na PSN (Personal Short Number), které je předem specifikováno mobilním operátorem.
- A obdrží potvrzovací SMS zprávu specifikující úspěšné založení skupiny. Rovněž tato zpráva obsahuje PSN, které je přiřazeno uživateli A jako reference k nově vytvořené skupině.
- A pozve do skupiny dalšího člena (B) pomocí SMS zprávy na přiřazené PSN, která je ve formátu příkazu, kterému Xtend rozumí.
- A obdrží potvrzovací zprávu o tom, že pozval do skupiny uživatele B.
- B obdrží zprávu týkající se pozvánky do skupiny. Může pozvánku přijmout nebo odmítnout, což udělá odpovědí na PSN, ze kterého pozvánka přišla.
- Jestliže B přijme pozvánku, obdrží potvrzovací SMS.
- Po obdržení potvrzovací SMS může na dané PSN zasílat zprávy a ty budou doručeny všem členům skupiny.

Aplikační server Group Messagingu obsahuje celou logiku služby a rozumí pouze předem definovaným příkazům, které si podle svých požadavků definuje mobilní operátor.

4.3 Vnitřní struktura aplikačního serveru

Xtend funguje jako cloud služba, která běží na virtuálních strojích v datacentrech, která si zákazníci platí. Xura využívá čtyři cloudová řešení, mezi která patří Amazon, NTT, NaviSite a OpenNebula. Jako operační systém je používán Red Hat Enterprise Linux.

Na serveru je implementováno mnoho služeb, mezi kterými musí být rozlišováno. O toto rozlišení se stará kód služby, což je předčíslí, které uživatelé dávají před telefonní číslo. SMSC podle tohoto kódu směruje zprávu na Xtend a ten podle kódu rozliší, o kterou službu jde. Každá ze služeb má svůj konfigurační soubor, který obsahuje informace o tom, jak se má daná služba chovat v určitých situacích.

V tomto konfiguračním souboru je definováno chování každé služby. Je to vyšší, abstraktní úroveň nad samotným kompilovaným kódem, který se provádí v pozadí. V první fázi jsou uvedeny deklarace s definicemi proměnných, které služba načítá při inicializaci. Nejdůležitější je kód služby, stavy a události. Syntaxe definice stavu nebo události je:

$$states/events = state/event : x$$

x udává číslici, která reprezentuje daný stav nebo událost.

Služby interně pracují jako konečný automat, kdy má každá služba definované stavy a události, které reprezentují přechody. V konfiguračním souboru je definice tohoto konečného automatu. Definice je zapsána ve formátu:

$$trans = state : event : state : callback$$

Přechod z aktuálního stavu do jiného stavu je definován tak, že se určí aktuální stav, událost, která nastane, následující stav, do kterého se konečný automat dostane po provedení akce a callback, což je jméno dané akce, která se při přechodu provede. Tímto způsobem je definován celý konečný automat, což tvoří danou službu. V pozadí je za každou z akcí reálný kus programu, který vykonává úlohy.

Dalšími povolenými konstrukcemi jsou přiřazení a klauzule *if*, která má následující syntaxi:

$$action = If : variable : is/is\_not : value : callback$$

Provedení je obdobné jako ve většině programovacích jazyků. Jestliže se proměnná rovná dané hodnotě, proveď akci. Přiřazení vypadá následovně:

$$action = Set\_Param : variable : value$$

Přiřazení je identifikováno klíčovým slovem *Set_Param*, po kterém následuje proměnná a hodnota, kterou má obsahovat. Mezi další podporované akce patří *Get_Next_State*, *Set_Next_State*, *Add*, *Create_Service_Msg*, *Set_Timer*, *Clear_Timer*, *Wait_For_User_Resp* a další, které mají ale širší kontext a pouhé zmínění by nebylo pro pochopení uspokojující.

Rovněž je možné skládat jednotlivé služby za sebe. To znamená, z jedné služby se plynule přepnout do služby jiné. Například pokud chce uživatel využít služby Group Messaging a nemá dostatek kreditu, může se automaticky iniciovat služba Collect SMS. Když se zákazník rozhodne, že chce nasadit novou službu, která ještě není implementovaná, díky tomuto systému se výrazně urychlí vývoj této nové služby. V jednodušších případech lze naimplementovat celou novou službu jen v systému tohoto konfiguračního souboru. Ovšem za předpokladu, že potřebné funkce v reálném kódu jsou skutečně všechny naprogramovány. Pokud by nějaké akce chyběly, musely by se dodatečně naprogramovat, až poté je lze tady využít.

Kapitola 5

Návrh řešení monitorovacího nástroje

Jak již bylo řečeno, tato bakalářská práce se zabývá řešením monitoringu aplikačního serveru Xtend. V této kapitole budou rozebrány primární i sekundární požadavky na aplikaci stejně jako požadavky, které byly stanoveny a které jsou už nad rámec této bakalářské práce. V sekci 5.1 bude vysvětleno, co za vstupní data bude program přijímat a které položky spadají do oblasti zájmu. Sekce 5.2 popisuje pohled ze strany klienta, co by měl být klient schopen vidět a jakým způsobem. V sekci 5.3 bude nastíněna problematika programu jako celku, jakým způsobem je potřeba zpracovávat data a rovněž návrh databáze, který je nutný implementovat. Poslední sekce kapitoly 5.4 popisuje způsob, jakým bude probíhat testování, kdo a jak bude testovat a v jakém prostředí.

Ještě před vstupem do dalšího tématu si je potřeba ujasnit, kdo bude koncovým uživatelem aplikace a jakým způsobem ji bude používat. Požadavky jsou takové, že by aplikace měla sloužit zákaznické podpoře, která řeší problémy zákazníků, jakožto koncových uživatelů. Mezi problémy se řadí například nejběžnější případ nedoručených zpráv nebo špatná platba za zprávy. To je ovšem ten nejhorší scénář, pokud chybu nahlašuje koncový zákazník mobilního operátora. Ještě před vydáním nové služby probíhá rozsáhlé testování jak služby, která se přidává, tak všech stávajících služeb, abychom zjistili, jestli nová služba nějakým negativním způsobem neovlivnila služby stávající. Takže se tento nástroj bude používat právě v testovacích fázích před vydáním nové služby.

5.1 Vstupní data a objekty zájmu

Požadavek na vstupní data je takový, aby byla aplikace kompatibilní s výstupními soubory aplikačního serveru. Xtend produkuje logovací soubory ve formátu CSV (*comma-separated values*), ze kterých aplikace vychází. Xtend u každé služby funguje tak, že se jakákoli akce, která se stane (přijetí/odeslání zprávy nebo přechod z jednoho stavu konečného automatu do dalšího), loguje na jeden řádek CSV logu. Formát CSV souboru je velmi jednoduchý. Na každém řádku jsou informace odděleny čárkou. Příklad takového souboru může být:

123,položka1,456,položka2,789,položka3

Jak lze vidět na příkladu, na řádku můžou být jakékoli informace, všechny jsou odděleny čárkou.

Každý řádek má tedy určitou vypovídající hodnotu o každé zprávě, ať už způsobila změnu v konečném automatu nebo byla přijata/odeslána, či proběhla jiná akce. První řádek

logovacího souboru obsahuje informace o pořadí jednotlivých položek. Například název virtuálního stroje, číslo dialogu, typ, stav. Právě z těchto dat celá aplikace vychází. Záleží na konfiguraci Xtendu u klienta, ale ve většině případů je nastavený tak, aby produkoval tento logovací soubor každou minutu.

Pro pracovníka zákaznické podpory jsou nejdůležitější tyto položky každého řádku:

- instance - název virtuálního stroje, který daný logovací soubor vyprodukoval.
- dialogue_id - číselná hodnota vytvořená podle aktuálního časového razítka, označující unikátní identifikátor dialogu.
- dialogue_type - číselná hodnota, reprezentující aktivovanou službu.
- event_type - číselná hodnota, reprezentující právě nastalou událost.
- dialogue_state - číselná hodnota, reprezentující nový stav, do kterého se dialog dostal po příchodu události.
- date - datum, kdy nastala nějaká akce.
- time - čas nastalé události.
- dialogue_short_code - kód služby, pomocí kterého se komunikuje se zákazníkem.
- dialogue_initiator - MSISDN strany, která aktivovala službu.
- dialogue_receiver - MSISDN strany, která hraje roli příjemce.
- text - zpráva v čitelné textové podobě.

Tato data aplikace extrahuje a dále zpracovává a ukládá do databáze.

5.2 Klientská prezentace

Všechny požadavky byly konzultovány s vedoucím oddělení zákaznické podpory týmu Xtend. Vysokoúrovňové požadavky jsou takové, aby když uživatel aplikace (pracovník zákaznické podpory) zadá jako parametr MSISDN (*Mobile Station International Subscriber Directory Number*), měla by mu aplikace vrátit kompletní seznam všech dialogů, kterých se dané telefonní číslo účastnilo jako adresát nebo odesílatel. Celá prezentace údajů je v rozhraní příkazové řádky. Přehledně by měl uživatel vidět, jak dané dialogy šly jednotlivými službami, například *Group Messaging -> Collect SMS*. Pod každým dialogem uvidí seznam vlastností, které si dialog nese. Mezi ně patří:

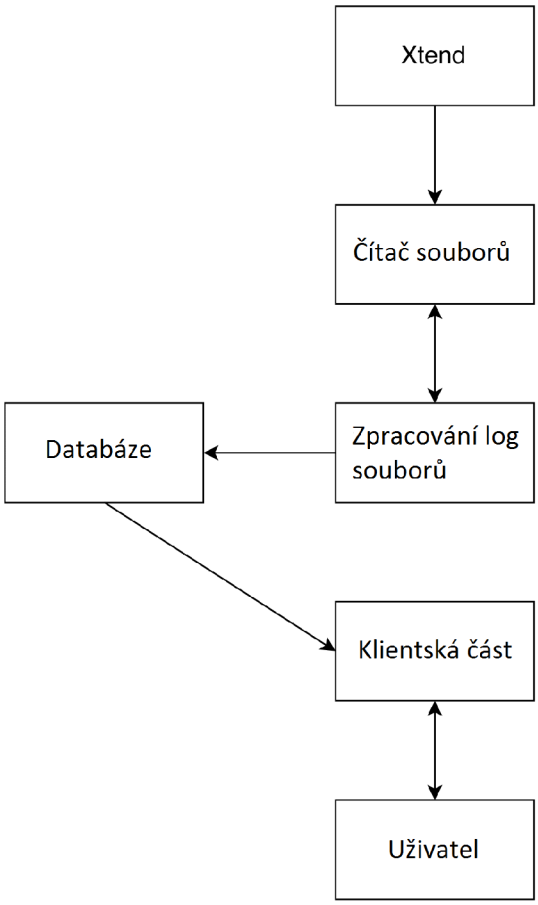
- Jméno služby.
- Strana A.
- Strana B.
- Kód služby.
- Čas počátku dialogu.
- Původní text zprávy.

Každý takový dialog by měl dále obsahovat aktuální stav konečného automatu služby, ve kterém se aktuálně nachází spolu se všemi stavy, kterými doposud prošel. Jak již bylo naznačeno v předchozí kapitole, stavy konečného automatu se definují pomocí číslic. Právě tyto číslice spolu s názvem stavu musí být zobrazeny u každého dialogu. Je to ta nejdůležitější informace, která pomůže při řešení nějakého problému. Uživatel aplikace ihned uvidí, ve kterém stavu se zpráva nacházela naposled a může posoudit, jestli je to stav, ve kterém vůbec měla být.

Další požadavky jsou nad rámec této práce a ačkoli v další verzi budou přístupné, tato práce je oficiálně nezahrnuje. Hlavním takovým požadavkem je grafické uživatelské rozhraní, které se bude nacházet na produktových webových stránkách Xury týmu Xtend nebo podpora ukončení stále aktivních dialogů, aby se uvolnily zdroje, které alokuje daná zpráva, jež už ovšem nemůže být z nějakého důvodu (chyby na straně serveru) doručena.

5.3 Schématický návrh řešení

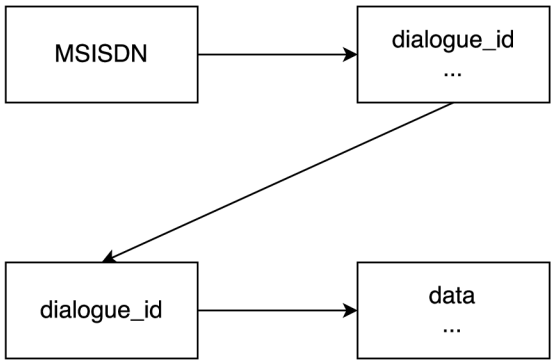
V následujícím schématu 5.1 bude vysokoúrovňově popsána struktura celé aplikace a jak spolupracuje se serverem Xtend.



Obrázek 5.1: Schéma aplikace

Struktura se skládá ze šesti částí. V první části, nahoře v diagramu, se nachází server Xtend, který produkuje logovací soubory. Ty načítá první část aplikace, čítač CSV souborů. Ten soubory načítá a data dále předává druhé části aplikace, která je zpracovává. Tento modul by měl extrahovat pouze důležité informace zmíněné v sekci 5.1 a ty uložit do databáze, což je samostatná entita. V databázi nemusí být uložena pouze perzistentní data, to záleží na konfiguraci. V dalším odstavci budou blíže rozebrány požadavky na databázové schéma. S databází komunikuje třetí část aplikace, což je část obsluhující požadavky klienta. Uživatel aplikace zasílá této části požadavky na data, ta si je vyžádá z databáze, zpracuje do čitelné podoby a vrátí uživateli.

V dalším schématu 5.2 je obecně znázorněna struktura databáze tak, jak by byla ideálně požadována.



Obrázek 5.2: Struktura databáze

Databáze by se měla skládat ze čtyř entit. První entitou je MSISDN, což je telefonní číslo ať už příjemce nebo odesílatele. Toto telefonní číslo se účastnilo pravděpodobně (minimálně v testovacích scénářích) mnoha dialogů. Každý dialog má svůj jedinečný číselný identifikátor *dialogue_id*. Právě s těmito identifikátory je MSISDN spřaženo. *dialogue_id* poté ukazuje na užitečná data. V relační databázi by to znamenalo mít dvě tabulky, kde v první by bylo MSISDN primárním klíčem s dalšími položkami *dialogue_id* a v druhé tabulce by bylo *dialogue_id* cizím klíčem spolu s daty. Zvolení samotné databáze a její vnitřní struktury bude řešeno v kapitole 6.

5.4 Testování aplikace

Za účelem prezentace rozhraní příkazové řádky je potřeba navrhnout klientské řešení, které bude z databáze brát požadovaná data a ty přehledně zobrazovat uživateli. Testování aplikace bude v první fázi probíhat ve spolupráci s pracovníky zákaznické podpory, hlavně z toho důvodu, že ti budou především aplikaci používat. Budou jim pokládány otázky na aplikaci jako takovou, jestli se jim s ní dobře pracuje, jestli je uživatelsky přívětivá a jednoduchá na pochopení. V další fázi bude produkt testován automatizovaně, kdy testovací mechanismus porovná data z aplikace s očekávanými daty.

Kapitola 6

Implementace

V této kapitole bakalářské práce bude rozebrána implementace celé aplikace, tzn. všech částí, ze kterých se skládá. V první sekci, databázové čáti 6.1, bude nejdříve vysvětlen rozdíl mezi relačním a nerelačním modelem dat a poté bude kladen důraz na typické zástupce relačních a nerelačních databází. Konkrétně budou zmíněny MySQL, PostgreSQL, MongoDB a Redis. Další sekce 6.2 bude zaměřena na serverovou část aplikace. V první části sekce bude vysvětleno, proč je aplikace psána v jazyku C++, další části popisují použité technologie a moduly, které jsou důležitými součástmi aplikace. Mezi ně patří třída Networking, RedisManager a modul Kontrolér. V poslední sekci 6.3 bude rozebrána část aplikace na straně klienta. Nejdříve bude naznačeno, proč byl pro klientskou stranu použit jazyk Python a dále bude popsán způsob získávání dat z databáze a překlad na srozumitelný formát, který může člověk jednoduše přečíst a pochopit a na základě toho řešit vzniklé problémy.

6.1 Databázová část

Databázová část aplikací je mnohdy kritická a může být úzkým hrdlem mnoha projektů, jedním z nich je tato práce. V sekci 6.1.6 bude vysvětleno, z jakého důvodu je nutné udělat správný výběr databáze se správnou strukturou dat. Tato kapitola se bude tedy problematikou databází zabývat více do hloubky. Popíše se nejprve databáze z obecného hlediska, co znamená pojem databáze, jaké typy databází se v praxi používají, relační a nerelační model dat, typické zástupce, které se ve firmě používají a využití databáze v konkrétním případě této aplikace.

Databáze je organizovaná, strojově čitelná kolekce dat. Je to kolekce schémat, tabulek, dotazů, pohledů a jiných objektů. Data jsou typicky organizována jako model aspektů z reality, pomocí těchto dat se snažíme vyjádřit objekty reálného světa.

Systém řízení báze dat

Systém řízení báze dat (SŘBD), což už název napovídá, je kus nějakého softwaru pro řízení databáze a přístupu k datům. V praxi se často používá pojem SŘBD jako synonymum k databázi samotné, obzvláště v komerční literatuře.

SŘBD odpovídá na dotazy, které ji zasílají aplikační programy ostatních uživatelů, kteří si přejí databázi používat. Příkazy jsou psány v databázovém jazyku SŘBD (např. SQL). Odpověď zahrnuje většinou návratový kód operace, zprávu a výsledek dotazu. Aby

databáze mohla podporovat operace mnoha uživatelů konkurentně, SŘBD pracuje jako serverová aplikace. Jakýkoli požadavek od aplikace, který si žádá pracovat s databází bude ihned obsloužen, protože je s ním z pohledu databáze zacházeno jako s klientem, který se k databázi připojí. Toto chování samozřejmě vyžaduje určitý komunikační protokol, pomocí kterého se klient (aplikace) připojí a zahájí sezení se serverem (SŘBD), poté si můžou vyměňovat data [1].

Aby mohla databáze odpovídat aplikačním programům, obvykle očekáváme, že SŘBD splňuje určité vlastnosti. Mezi ně patří například:

- Vytvořit, aktualizovat a zrušit proměnnou v databázi.
- Přijmout a zpracovat integritní omezení.
- Dovolovat autorizaci uživatelů při přístupu k datům.
- Poskytnout výsledek dotazu.

Databázový jazyk

Jak bylo řečeno, příkazy směrované databázi od ostatních aplikací jsou napsané v databázovém jazyku, který daná SŘBD podporuje a může je tedy vykonat. Někdy je namísto databázového jazyku používán termín *data sublanguage*. Předpona *sub* odkazuje na fakt, že aplikační programy jsou často psané v programovacích jazycích pro obecné použití (hostovací jazyky), ve kterých jsou zapouzdřeny databázové příkazy. Někdy tento způsob zapouzdření příkazů nefunguje zcela přesně, protože kompilér nebo interpret hostovacího jazyka nemusí rozumět těmto zapouzdřeným příkazům. V tom případě se využívá preprocesoru, který nahradí zapouzdřené příkazy jinými, srozumitelnými SŘBD.

Databázové dotazy jsou výrazy, které při vyhodnocení vrací výsledek zderivovaný z databáze. Dotazy jsou "nástroje", ta podstatná část, která databázi činí použitelnou. Databázový dotaz sám o sobě není příkaz. Ve skutečnosti SŘBD podporuje příkazy, jejichž provedení spěje k vykonání dotazu. Po vykonání dotazu SŘBD vrací výsledek aplikaci, která byla odesílatelem dotazu. V praxi aplikační program zasílá databázi dotazy, které vrací výsledky, jenž aplikace zobrazuje například v tabulkové formě [1].

6.1.1 Relační a nerelační model dat

Model dat

Model dat je nejfundamentálnějším pojmem v oblasti databázových systémů. Model dat je notace pro popis dat nebo informace. Ta se skládá ze tří částí.

První je struktura dat. Lze zde užít přirovnání k programovacím jazykům jako například C nebo Java pro popis struktur dat použitých v programu. Takovými datovými strukturami může být například datový typ pole, struktura nebo objekt. Datové struktury použity pro implementaci dat v databázových systémech jsou často v praxi zmiňovány jako fyzický model dat, ačkoli je to stále vysoce odstíněno od samotného hardwaru. V databázovém světě je model dat něco na vyšší úrovni než datové struktury a je často nazýván konceptuální model.

Druhou částí jsou operace nad daty. V programovacích jazycích je operace nad daty obecně něco, co může být naprogramováno. V databázových datových modelech je to obvykle omezená sada operací, jež můžeme využívat. Jsme schopni vykonávat omezený počet

dotazů (databázová operace na získání dat) a modifikací (operace na změnu dat). Tato limitace v počtu operací ovšem není slabinou, ale naopak výhodou. Tím, že limitujeme proveditelné operace jsme schopni jako programátoři popsat databázové operace na velmi vysoké úrovni a zároveň SŘBD tyto operace implementuje velmi efektivně.

Třetí částí jsou datová omezení. Databázové modely obvykle mají způsob pro popis limitací, která data musí dodržovat. Tyto limitace můžou být primitivní (den v týdnu musí být celočíselnou kladnou hodnotou mezi 1 a 7) až po velmi komplexní (*assert* nebo *trigger*) [3].

Relační model dat

Relační databáze je jeden ze způsobů, jak organizovat data do kolekce relací. Pokud bychom chtěli pojem relační databáze vizualizovat, použili bychom například tabulku 6.1.

StudentID	**Name**	**CourseID**
123	Thomas	1
456	Lukas	2
789	John	3
159	Veronica	4
357	Oliver	5

Tabulka 6.1: Relace zobrazena v tabulkové formě.

První řádek je hlavička a každá z položek hlavičky se nazývá atribut. Následující řádky jsou n-tice o stupni 3 (počet hodnot v jednom řádku), což je stupeň relace. Všechny tyto n-tice tvoří tělo. Dohromady je jich v této tabulce 5, což je kardinalita relace. Ačkoli bude dále vysvětlen rozdíl mezi relací a tabulkou, v praxi se používá asociace:

- relace: tabulka
- n-tice: řádek
- atribut: sloupec

Vizuální reprezentace tabulky, která zde byla použita se hodí zejména na popis relačních databází natolik, že mnoho lidí používá slovo *tabulka* jako alternativu k pojmu *relace*. Jazyk SQL v podstatě tento výraz používá, takže v kontextu relační teorie je vhodné se zastavit na pojmu relace v teoretické rovině, abychom mohli zvýraznit rozdíly mezi tabulkami a relacemi.

Relace je formální termín v matematice predikátové logiky [16]:

- n-ární relací mezi množinami A_1, A_2, A_3, ...,A_n, kde $n \in N$, rozumíme libovolnou podmnožinu kartézského součinu n množin.
- n-ární relací na množině A je tedy libovolná množina R uspořádaných n-tic, přičemž $R \subseteq A^n$.

Relace lze rozdělit podle počtu množin kartézského součinu náledovně:

- Unární relací nazveme každou podmnožinu množiny M.

- Binární relací nazveme každou množinu uspořádaných dvojic $[x; y] \in M^2$.
- Ternární relací nazveme každou množinu uspořádaných trojic $[x; y; z] \in M^3$.
- Ostatní relace jsou označovány buď souhrnným názvem n-ární relace nebo konkrétně podle vzoru: kvartární, pentární, sextární, septární, oktární, nonární atp.

Apeluje se zde na pojem vztahu mezi "věcmi", v matematickém světě mezi množinami prvků. Většina matematických textů se zaměřuje na relace párů, ovšem naše případy užití ukazují, že nás zajímá spíše jakýkoli počet prvků, tedy zcela obecné použití.

Relace se ovšem nerovná tabulka. Tabulkou je zde myšlena například tabulka 6.1. Pojem relace a tabulka nejsou synonyma, protože ačkoli každá relace lze znázornit tabulkou, ne každá tabulka je reprezentací té samé relace. Navíc mnoho různých tabulek může reprezentovat tu samou relaci. Kdybychom v tabulce 6.1 například změnili pořadí sloupců a pořadí řádků, ovšem hodnoty bychom nechali původní, pořád by tato tabulka byla definována tou samou relací jako tabulka původní. Jde o ilustraci toho, že relace reprezentovaná tabulkou nezávisí na pořadí sloupců nebo řádků. Mnoho takových modifikovaných tabulek se vztahuje k té samé původní relaci, neboť můžeme libovolně měnit pořadí sloupců zleva doprava a pořadí řádků shora dolů.

Jak již bylo zmíněno, relační model dat je založen na tabulkách. Struktura relačního modelu by šla přirovnat v programovacím světě například na pole struktur v jazyku C, kde atribut je jméno položky a každý řádek v tabulce reprezentuje instanci struktury v poli. Musí být ovšem podotknuto, že fyzická implementace databázových struktur ani nemůže být jiná [3].

Nerelační model dat

Nerelační model dat je charakteristický pro databáze, které nepoužívají SQL, tzv. NoSQL databáze. Některé sice používají dotazovací jazyk, například *Cassandra*, nedosahuje ovšem takové flexibility jako u relačních databází. Ačkoli je termín NoSQL často spojován s proprietárními systémy, je to open-source fenomén. Mnoho NoSQL databází jsou určené k fungování na clusterech. To má především efekt na model dat a přístup k vlastnosti konzistence. Relační databáze používají ACID transakce na řízení konzistence v celé databázi. To se neodmyslitelně střetává s cluster prostředím, kde NoSQL databáze nabízí rovněž možnosti pro konzistenci a distribuci dat.

NoSQL databáze jsou zasazeny do 21. století, kdy jsou kladeny čím dál větší nároky převážně na webové aplikace. Takové databáze pracují bez striktního schématu, což dovoluje přidávat nové informace do databáze bez nutnosti měnit strukturu dat. Dnes se hojně používá pojem *polyglot persistence*, což znamená využití různých způsobů jak ukládat data v závislosti na okolnostech, místo prostého zvolení relační databáze jako tomu bylo v minulém století.

Dále budou blíže popsány Key-Value databáze, Document databáze a Column-Family uložiště.

Key-Value databáze jsou založeny na hash funkci, primárně při přístupech do databáze přes klíč. Analogicky si lze tento typ databáze představit jako databázi relační se dvěma sloupci - *IDENTIFIKÁTOR* a *HODNOTA*. Identifikátor zde plní úlohu klíče a hodnota plní úlohu uložených dat. Pokud záznam s daným identifikátorem v databázi již existuje, bude hodnota přepsána novou, v opačném případě se vytvoří nový záznam. Tento typ ukládání dat je z hlediska aplikačního programového rozhraní nejjednodušší na poli NoSQL databází.

Uživatel potřebuje tři základní operace - vložení, čtení a smazání záznamu. Tyto databáze patří mezi nejvýkonější a jsou lehce škálovatelné. Známými zástupci jsou například Riak, Redis nebo Berkeley DB.

Document databáze zaládají svoji funkčnost na dokumentech. Databáze tohoto typu ukládají a žádají dokumenty formátu například XML, JSON nebo BSON. Tyto dokumenty jsou sebepopisující datové struktury hierarchického stromu, které se skládají z map, kolekcí a skalárních hodnot. Uložené dokumenty vypadají velmi podobně, ale nejsou přesně stejné. Document databáze ukládají dokumenty v části *hodnota* Key-Value uložiště. S tímto typem databáze se pracuje stejným principem jako s Key-Value databází. Mezi známé zástupce patří MongoDB, MarkLogic nebo CouchDB.

Column-Family uložiště dovolují ukládat data s klíči namapovanými na hodnoty nebo na spojení hodnot do více sloupcových rodin. V tomto případě je každý sloupec rodiny mapa dat. V praxi si lze tento formát představit jako spojení principu relační databáze s Key-Value databází. Každý řádek jednoznačně identifikuje nějaký identifikátor a každý z ostatních sloupců obsahuje vzor *IDENTIFIKÁTOR* a *HODNOTA*. Takový řádek je potom Column-Family. Známými zástupci jsou například Cassandra, HyperTable nebo Amazon SimpleDB [5].

6.1.2 MySQL

MySQL je open-source relační databáze. Zdrojové kódy jsou k dispozici pod licencí GNU General Public License. Pro proprietární užití nabízí placené edice s funkcionalitou navíc. MySQL je centrální komponenta open-source webového aplikačního stacku LAMP. LAMP je akronym pro *Linux, Apache, MySQL, Perl/PHP/Python*. Je to jedna z nejčastěji využívaných databází převážně díky její jednoduchosti konfigurace a obecné práci s ní. Aktuálně patří tato databáze pod Oracle. Je napsaná v programovacím jazyce C/C++. Podporovanými platformami jsou Windows, Linux, Solaris, Mac OS X a FreeBSD [12].

6.1.3 PostgreSQL

PostgreSQL, často přezdívaná Postgres je objektově-relační databáze, s možností použít objektového návrhu s důrazem na rozšiřitelnost a striktního dodržení standardu. Stejně jako MySQL je open-source a využívá vlastního relačního schématu, rovněž je jednoduchá na konfiguraci a je vysoce škálovatelná. Je schopna pojmout zátěž od malých aplikací běžících na jednom stroji až po velké světově známé aplikace s mnoha konkurentními uživateli a s obrovskými datovými uložištěmi. Na Mac OS X je PostgreSQL výchozí databází, dále je možné ji použít na systémech Windows a Linux. Je napsaná v programovacím jazyku C [14].

6.1.4 MongoDB

MongoDB je open-source document-oriented databáze. Patří mezi NoSQL databáze. Používá pro ukládání hodnot JSON dokumenty, v tom spočívá velká flexibilita v možnosti ukládání dat. MongoDB je vyvíjena firmou MongoDB Inc., je zdarma pod licencí GNU Affero General Public Licence a Apache Licence. Díky tomu, že je tato databáze orientována na JSON dokumenty, vývoj i integrace aplikací je velmi urychlena a usnadněna. Jako programovací jazyky byly využity C/C++ a JavaScript. Mezi hlavní funkci patří *Ad-hoc dotazy*, které podporují široký výběr dotazů a regulárních výrazů. Tyto dotazy mohou rovněž obsahovat uživatelem definované JavaScript funkce [11].

6.1.5 Redis

Redis (Remote Dictionary Server) je open-source softwarový projekt sponzorován Redis Labs, který implementuje NoSQL serverovou databázi. Redis podporuje spojení se sítěmi, pracuje jako in-memory databáze, což znamená, že záznamy ukládá do paměti a dále volitelně na disk jako perzistenci. Redis je konkrétně Key-Value NoSQL databáze. Je napsán v programovacím jazyce C. Ačkoli Redis uchovává data v paměti, jsou zde dvě možnosti perzistence. První je snapshotting (RDB), který v určitých časových intervalech zapíše všechna data z paměti na disk. Druhou, bezpečnější variantou vůči ztrátě dat, je žurnálování (AOF). Tato varianta využívá souboru, do kterého se připisují změny, zatímco je tento soubor hlídán proti nekonečnému růstu [15].

6.1.6 Volba vhodné databáze

Jako databázovou část pro tento projekt jsem zvolil Redis. Důvody jsou pro toto rozhodnutí následující. Redis je z výše zmíněných databází nejvýkonnější, položky má uložené pouze v paměti a perzistence je volitelná. Monitorovací aplikace spolu s databází běží na stejném fyzickém stroji jako aplikační server Xtend. Po výkonnostním testování performance testerů bylo zjištěno, že pevné disky jsou největší slabinou při běhu Xtendu. Xtend používá rovněž Redis, což je dalším důvodem pro volbu této databáze. Ovšem Xtend vyžaduje perzistenci dat ve formě RDB režimu. Při velkém provozu, který Xtend zpracovává Redis zapisuje na disk velmi často mnoho dat. Při testování krajních případů několikrát nastala situace, kdy se virtuální stroj nečekaně zhroutil z důvodu zahlcení na straně pevných disků. Monitorovací aplikace perzistenci dat nevyžaduje, neboť je schopná z CSV souborů poskládat po výpadku stroje celou databázi znovu. S výkonem monitorovacího nástroje a Redisu na testovacím virtuálním stroji je to při velikosti 1GiB zdrojových CSV souborů otázka dvaceti minut.

Vnitřní struktura Redisu bude v případě monitorovacího nástroje následovat návrh z obrázku 5.2. V databázi budou dva druhy záznamu typu Key-Value. V prvním typu záznamu bude klíčem MSISDN (odesílatele či příjemce) a value bude datový typ *SET*, do kterého se budou ukládat *dialogue_id*. Datový typ *SET* si lze představit jako pole typu řetězec, přičemž položka do tohoto pole lze uložit za předpokladu, že zatím neexistuje. Časová složitost operací nad tímto datovým typem je lineární. Ve druhém typu záznamu bude klíčem *dialogue_id* a value bude datový typ *LIST*. Tento datový typ byl zvolen z důvodu chronologického ukládání. Je potřeba, aby záznamy byly uloženy za sebou. *SET* je v tomto případě nevyhovující, neboť vnitřně nemusí ukládat položky nutně za sebe. Položkou typu *LIST* jsou již samotná data, která jsou důležitá pro koncového uživatele. V konfiguračním souboru aplikace je dále uvedena IP adresa a port instance Redisu a čas expirace dat.

6.2 Serverová část

V této sekci bude popsána vnitřní implementace monitorovací aplikace. V sekci 6.2.1 bude vysvětleno, proč jsem zvolil programovací jazyk C++ spolu s jeho krátkou charakteristikou. V další sekci 6.2.2 bude objasněna implementace třídy *Networking*, která slouží pro základní operování se sockety. Další sekce 6.2.3 popisuje implementaci třídy *RedisManager*, která vytváří rozhraní pro práci s Redis databází na aplikační úrovni. Poslední sekce 6.2.4 je věnována popisu implementace klíčového modulu kontroléru, který pro svou funkčnost využívá aplikačního rozhraní ostatních tříd. Tento modul je jádrem aplikace. Stará se o načítání a zpracování souborů s následným uložením do databáze.

6.2.1 C++

C++ je programovací jazyk pro obecné použití. Má vlastnosti imperativního, objektově orientovaného a generického programování, přičemž podporuje nízkoúrovňovou práci s pamětí. Byl designován pro systémové programování, programování vestavěných systémů s omezenou pamětí a tvorbu aplikací kritických na výkon. Používá se převážně pro desktopové (webové vyhledávače, hry) a serverové aplikace (e-commerce, SQL servery). C++ je kompilovaný jazyk s implementací zahrnující Free Software Foundation, LLVM, Microsoft, Intel a IBM.

Je standardizován organizací ISO (International Organization for Standardization) naposledy roku 2014 (C++14), původně standardizován roku 1998 (C++03). Před tím, než byl poprvé standardizován, byl vyvíjen Bjarnem Stroustrupem v Bell Labs jako rozšíření jazyka C. Na konci roku 2017 bude vydán další standard (C++17), jehož nové vlastnosti už má mnoho kompilérů implementovaných. Další standard se plánuje na rok 2020 (C++20). Tímto jazykem bylo ovlivněno mnoho jiných jazyků, které přišly poté (C#, Java). Mezi nejznámější implementace kompilérů patří LLVM Clang, GCC, Microsoft Visual C++ a Intel C++ Compiler [10].

Jazyk C++ byl zvolen převážně z toho důvodu, že se používá pro celý produkt Xtend. Aplikační server Xtend je psán v jazyku C++ a tudíž by bylo nesmyslné a značně komplikované snažit se vytvořit kompatibilní vrstvu mezi C++ a jiným jazykem. Dalším důvodem je udržovatelnost, neboť všichni členové týmu píšící serverové části používají C++, popřípadě Python na části, které nejsou kritické na výkon.

6.2.2 Třída Networking

Tato třída, kterou jsem pro účely práce vytvořil, implementuje základní operace nad BSD sockety na straně serveru. Byla vytvořena za účelem možnosti připojení vícero klientů k procesu monitorovací aplikace. Prvotní podnět k implementaci třídy s touto funkcionalitou bylo potřeba vzdáleně ukončit aplikaci. To je umožněno tak, že klient zašle procesu zprávu s textem *shutdown* a aplikace skončí. Ačkoli k tomuto rozhodnutí vedl pouze jeden požadavek, aplikační programové rozhraní (API) této třídy dovoluje jednoduše přidat do kontroléru další funkcionalitu založenou na práci se socketama. Dále bude toto API podrobněji rozebráno. Postupovat se bude logicky od konstruktoru, přes metody pro binding socketu až po operace extrahování přijatých dat.
Třída obsahuje tři konstruktory pro tvoření objektů, což jsou následující tři hlavičky:

- *TcpSocket() = default*
- *TcpSocket(const int&, const int&)*
- *TcpSocket(const int&&, const int&&)*

První konstruktor je defaultní, což je vlastnost jazyka C++, kdy tím dá programátor kompiléru najevo, že má vygenerovat konstruktor automaticky. Druhý konstruktor je tzv. copy konstruktor, který vyžaduje dva parametry typu reference na konstantní celočíselnou hodnotu a obsah těchto dvou parametrů zkopíruje v těle konstruktoru do svých privátních členských proměnných. Prvním parametrem je port a druhým parametrem je maximální počet připojených klientů v jednom čase. Třetí, tzv. move konstruktor je specifický tím, že očekává rvalue parametry, které jako v předchozím případě přiřadí členským proměnným objektu, ovšem parametry může zneplatnit. Tento konstruktor je výkonější než copy konstruktor, neboť se zde ušetří zbytečné kopírování dat v paměti.

Poté, co je vytvořena instance třídy, je důležité zavolat metodu s hlavičkou:

SocketState TcpSocket::setSocket()

Definice této metody obsahuje základní operace pro práci se sockety. Nejprve se detekuje, jestli daný socket již naslouchá na nějakém portu. Pokud ano, vrátí náležitý status. Jestliže ne, vytvoří se nový socket, nastaví se rodina *AF_INET*, adresa *INADDR_ANY* a port, na kterém bude socket naslouchat.

Pro nastavení naslouchání je potřeba zavolat metodu:

SocketState TcpSocket::setSocketListen()

Nejdříve se zavolá funkce bind na socket, který byl vytvořen ve volání předchozí metody. Následně se volá funkce listen, která zapřičiní, že socket je připraven na akceptování příchozích klientů.

Akceptování klientů probíhá v metodě:

SocketState TcpSocket::acceptConnection(TcpSocket)*

Parametrem je socket, který je nastaven aby naslouchal pro nová připojení. Když se klient snaží připojit, je uložena jeho identifikace.

Poslední důležitá metoda je:

SocketState TcpSocket::receiveData()

Ta slouží k obdržení dat od klienta. To je dosaženo funkcí *recv*, kdy se data ukládají do té doby, dokud se klient neodpojí, tedy dokud nezavře socket.

Metody vrací návratový kód ve formě *SocketState*, což je datový typ *enum class*. Ten se liší od klasického datového typu *enum* v jazyce C tím, že obsahuje prostor jmen, kterému všechny enumeráty náleží. Tudíž při direktivě *#include* nějakého modulu nemůže nastat křížení logicky různých enumerátů stejného jména. Výčet enumerátů je *Done*, *CannotSet*, *AlreadyListening*, *CannotBind*, *CannotListen*, *CannotAcceptClient* a *CannotReceiveData*.

Třída obsahuje další metody, které už nejsou tolik podstatné. Se sockety je manipulováno vícevláknově a asynchronně, tudíž žádná operace ve výsledku není blokující [2].

6.2.3 Třída RedisManager

Třída RedisManager, kterou jsem pro účely práce vytvořil, umožňuje práci s Redis databází a zapouzdřuje již existující knihovnu *hiredis* tak, aby podporovala objektově orientovaný přístup. Knihovna *hiredis* umožňuje práci s databází, ovšem nízkoúrovňově, neboť je psaná v jazyku C a neumožňuje objektový návrh. Třída poskytuje API, které je omezené pouze na požadavky monitorovací aplikace. Podporovat všechny operace, které Redis podporuje, by v tomto případě bylo zbytečné. Dále bude zjednodušeně nastíněna implementace tohoto API.

Třída obsahuje tři konstruktory pro tvoření objektů, které reprezentují následující tři hlavičky:

- *RedisManager::RedisManager()*
- *RedisManager::RedisManager(const std::string&, const int&)*
- *RedisManager::RedisManager(const std::string&&, const int&&)*

Konstruktory vypadají velmi podobně jako v třídě Networking s tím rozdílem, že druhý i třetí konstruktor vyžaduje první parametr typu reference na konstantní řetězec, což reprezentuje IP adresu instance databáze a druhý parametr typu reference na konstantní celočíselnou hodnotu, která zastupuje port, na kterém databáze naslouchá přicházející spojení. Rozdíl je rovněž v prvním konstruktoru, kde není uvedeno klíčové slovo *default*. Tato verze má vlastní definici defaultního konstruktoru, kde se jako IP adresa přiřadí *127.0.0.1* a port *6379*, což jsou standardní hodnoty v případě, že databáze běží na stejném systému jako monitorovací proces a původní konfigurace Redisu nebyla změněna.

Metody *SET* a *GET* není potřeba rozebírat. Jejich úlohou je pouze v případě *SET* nastavit novou hodnotu IP adresy, či novou hodnotu portu databáze, kam se má proces nadále připojovat a v případě *GET* tyto položky vrátit.

Poté, co je korektně vytvořená instance třídy, může být použita pro zasílání příkazů systému řízení báze dat na straně Redisu. K tomu slouží metoda s hlavičkou:

void RedisManager::executeCommand(const std::string&)

Tato metoda vyžaduje pouze jeden parametr typu reference na konstantní řetězec. V tomto řetězci je uložen celý příkaz, který bude zaslán databázi. Samotné zaslání se ovšem v této metodě ještě nekoná. Postup je několikaúrovňový, aby v případě nalezení chyby bylo jednoznačné, kde se daná chyba nachází. Další důvod je oddělit zpracování dotazu před samotným zasláním. V této metodě se detekuje první slovo, první část dotazu, konkrétně operace typu *SET* nebo *GET*. Metoda podle typu operace invokuje privátní metody:

- *void processGetOperation(const std::string&)*
- *void processSetOperation(const std::string&, const std::string&)*

Privátní metoda *Get* je jednoduchá, pouze zašle dotaz databázi. V jediném parametru obdrží dotaz, který databázi přepošle a další operace nekoná. Privátní metoda *Set* očekává dva parametry typu řetězec. První parametr obsahuje operaci, o kterou se jedná. Další parametr obsahuje zbytek dotazu, ze kterého se dále extrahuje část klíče. Pro korektní sestavení dotazu je zapotřebí mít všechny tři části odděleně - operace, klíč, hodnota.

Po vykonání operace je korektní postup vyžádat si z databáze návratový kód proběhlé operace. O to se stará metoda:

std::string RedisManager::getErrorMessage() const

Při úspěchu vrátí řetězec reprezentující, že operace proběhla úspěšně. V opačném případě vrací řetězec, který metoda dostala přímo z databáze jako návratovou zprávu o tom, co neproběhlo správně.

V případě, že operace proběhla korektně, a daná operace byla typu *GET*, je žádoucí obdržet výsledek operace. K tomu slouží metoda:

std::vector<std::string> RedisManager::getReply() const

Hiredis má vnitřně implementováno více návratových typů. Z toho důvodu je návratový typ této metody vektor řetězců, což je nadmnožina všech návratových typů hiredisu. Jejich výčet je:

- REDIS_REPLY_STATUS - obsahuje řetězec se statusem, jak skončila operace. Tato funkcionalita je implementována v metodě *getErrorMessage*.
- REDIS_REPLY_ERROR - má stejnou funkcionalitu jako REDIS_REPLY_STATUS.

- REDIS_REPLY_INTEGER - odpověď je ve formátu celočíselné hodnoty.
- REDIS_REPLY_NIL - nil bude vrácen, pokud nejsou k dispozici žádná data.
- REDIS_REPLY_STRING - řetězec, který obsahuje návratová data z databáze. Jedná se o jednu položku.
- REDIS_REPLY_ARRAY - odpověď, která obsahuje vícero řetězců jako datový typ pole.

Především kvůli poslednímu typu odpovědi je zvolen návratový datový typ metody vektor řetězců, aby mohlo být navráceno celé pole. V případě ostatních odpovědí bude mít vektor jen jednu položku. První a druhý typ odpovědi je implementován v metodě *getErrorMessage*, takže není potřeba podporovat tyto typy. Pokud bude navrácen nil, metoda vrátí prázdný vektor, jestliže integer, bude převeden na řetězec.

6.2.4 Kontrolér

Kontrolér není psán jako třída, nýbrž procedurálně jako modul. K tomuto rozhodnutí vedl fakt, že obsahuje funkcionalitu, která není znovupoužitelná a tudíž ji nemá smysl implementovat objektově. Pro svoji funkčnost naopak využívá aplikačního rozhraní ostatních tříd (Networking, RedisManager) a modulu SourceHandler. Tento modul je jádrem aplikace, řídí všechny úkony od načítání a zpracování souborů až po ukládání do databáze. Ačkoli modul obsahuje širokou funkcionalitu, bude v této sekci popsána jen ta klíčová, důležitá pro objasnění, jak modul funguje.

Aplikace funguje tím způsobem, že si z konfiguračního souboru načte adresář, který má monitorovat. Následně při vytvoření souboru v tomto adresáři, který je monitorován, dostane aplikace událost se jménem souboru, který může dále zpracovat. Tuto funkcionalitu umožňuje nástroj *inotify*, který je specifický pro linuxové systémy. Úlohu monitorování adresáře zaopatřuje modul SourceHandler. CSV soubory ke zpracování do monitorovaného adresáře doručuje speciální bash skript, který je kopíruje z jiných adresářů, to je závislé na konfiguraci u zákazníka.

Když *inotify* zaznamená přítomnost nového souboru, zavolá se funkce *processFile*, která požaduje parametr typu řetězce, což reprezentuje název souboru. Tato funkce řídí další zpracování souboru. Nejprve proběhne kontrola koncovky souboru a formátu, aby se rozhodlo, zdali je tento soubor validní a může být zpracován.

Následně je zavolána funkce *parseFile*, která jako první parametr požaduje název souboru a druhý parametr vektor řetězců pro uložení extrahovaných dat. V této funkci probíhá extrahování užitečných dat z každého řádku CSV souboru. Každý řádek obsahuje aktuálně dvacet položek, z nichž pro naše účely je jich důležitých pouze devět. V tomto případě nebylo potřeba *regex* funkcí, protože extrahovat položky ze souboru formátu CSV je velmi jednoduché, stačilo vytvořit algoritmus, který pomocí C++ funkcí *find* a *substr* extrahuje požadované položky. Funkce následně vrací vektor těchto položek jako řetězce.

Jestliže je tento vektor neprázdný, zavolá se funkce *processRedis* právě s parametrem tohoto vektoru. Zde se nachází kód pro databázovou část, konkrétně zpracování dat a uložení do Redisu. Nejdříve se vytvoří instance třídy RedisManager, skrze kterou je možné operovat s databází. Všemi položkami vektoru se iteruje a ukládají se do databáze pomocí specifických příkazů, které náleží daným datovým typům:

- *SADD* pro datový typ *SET*.

- *RPUSH* pro datový typ *LIST*.
- *EXPIRE* pro nastavení expirace dat.

EXPIRE nepatří ke specifickému datovému typu. Nastavuje se danému klíči po každém volání *SADD* nebo *RPUSH*, aby data nezůstávala v databázi zbytečně dlouho. Tato hodnota je konfigurovatelná, avšak výchozí hodnota je nastavena na jeden den. U každého zákazníka se to bude lišit podle požadavků požadavků pracovníků zákaznické podpory.

Hlavní smyčka programu běží, dokud kontrolní proměnná určující stav programu nenabude hodnoty *false*. Této hodnoty se může dosáhnout dvěma různými způsoby. Buď vzdáleně nebo lokálně. Vzdáleným ukončením procesu je myšleno připojení se na proces pomocí socketu například z nějaké aplikace (klientská část, curl) a zaslání příkazu *shutdown*. Lokálně to lze udělat velmi jednoduše příkazem:

kill -15 pid

pid zde označuje číslo procesu monitorovací aplikace, které lze získat pomocí příkazu:

ps aux | grep msg_monitor

Interně se spustí v obou případech (vzdáleně i lokálně) obsluha signálu *SIGTERM*, kde se invokuje funkce, která změní atomickou proměnnou na *false*. Přeruší se cyklus procesu a korektně se ukončí.

V případě ukončení vzdálenou cestou, tzn. přes socket, proces obsluhuje příchozí klienty konkurentním způsobem pomocí vláken. Pro každého nového klienta, který se připojí, je vytvořeno speciální vlákno, které jej obslouží a na základě dotazu vykoná požadovanou akci, v tomto případě ukončení procesu.

Nad samotným procesem leží další vrstva (OAM), která jej automaticky spouští, kontroluje běh procesu a může jej rovněž ukončit. Zároveň ukládá textový výstup procesu do logovacích souborů. V případě pádu procesu je touto vrstvou automaticky opětovně spuštěn. Monitorovací aplikace může být sestavena bez podpory této vrstvy a může pracovat zcela samostatně, což je zajištěno makrem v programu *make*. Tato speciální vrstva na kontrolu běžících procesů zde nebude dále rozebrána, neboť jde o priprietární systém a sahá nad rámec této bakalářské práce.

6.3 Uživatelská část

V této sekci bude detailně vysvětlena implementace klientské části a programovací jazyk Python, ve kterém je psána. V sekci 6.3.1 bude představen skriptovací jazyk Python, jeho použití a proč byl zvolen pro klientskou část aplikace. V části implementace klientské části bude nejdříve popsáno, jaké jsou povoleny argumenty skriptu, následně získání dat z databáze, jejich zpracování do čitelné podoby a následné mapování čísel na skutečné názvy položek.

6.3.1 Python

Python je jeden z nejpoužívanějších vysokoúrovňových programovacích jazyků pro obecné použití, poprvé vydán roku 1991. Python je interpretován a jeho filosofie je kladena na čitelnost kódu. To je docíleno především tím, že nutí programátora odsazovat kód, což je nedílnou součástí syntaxe Pythonu. Ten stejný algoritmus lze v Pythonu vyjádřit na výrazně nižším počtu řádků než například v jazycích jako C++ nebo Java. Jazyk je vhodný

pro psaní malých, velkých, konzolových i grafických programů. Python je dynamicky typovaný s automatickou správou paměti. Podporuje více programovacích paradigmat - objektově orientované, procedurální a funkcionální. Má rozsáhlou standardní knihovnu. Interpret Pythonu je dostupný na mnoha operačních systémech, mezi nejpoužívanější patří Linux, Windows a Mac OS X [4].

Tento jazyk byl použit pro implementaci klientské části z důvodu jeho lehké použitelnosti a rychlosti vývoje. Na virtuálních strojích firemního cloudu je všude Python předinstalován, takže nemusí být pochyby o tom, že by nebylo možné skript spustit. Rovněž je u Pythonu vysoká učící křivka, jde se jej naučit velmi rychle, což je další nezpochybnitelná výhoda. Dalším argumentem je výkon. Klientská část aplikace není výpočetně náročná, tudíž není důvod nepoužít skriptovací jazyk.

6.3.2 Implementace klientské části

Jak bylo řečeno v předchozí sekci, klientská část je psána v Pythonu a slouží k obsluze klientů, kteří chtějí monitorovací aplikaci použít. Její úlohou je vyhodnotit správnost vstupu od uživatele, následně zaslat dotaz databázi, která v případě kladného scénáře obratem vrátí data, která aplikace zpracuje, provede mapování hodnot na lidsky čitelné údaje a poté zformátuje a vypíše na standardní výstup. V této sekci bude blíže popsána implementace každé z těchto částí.

Uživatel, většinou pracovník zákaznické podpory, s aplikací komunikuje pomocí rozhraní příkazové řádky. Jsou tři korektní formáty spuštění skriptu:

- *python client.py clean*
- *python client.py shutdown*
- *python client.py MSISDN*

První způsob má za následek smazání celé databáze. Klient zašle přímo Redis databázi příkaz *FLUSHALL* a ta smaže z paměti všechny záznamy. Tato volba je vhodná při testování Xtend serveru, kdy už databáze zabírá mnoho paměti a záznamy jsou neaktuální nebo chybné.

Druhá možnost vede k vypnutí serverové strany aplikace. Jestliže se jako parametr použije *shutdown*, skript otevře socket a připojí se ne jako v prvním případě k databázi, ale k serveru. Ze sekce 6.2.4 víme, že server spustí obsluhu klienta v novém vlákně nezávislém na hlavním toku programu. Proběhne zpracování zprávy a po zjištění, že jde o příkaz *shutdown*, server dokončí aktuální zpracování souboru, pokud nějaký zpracovává, a ukončí smyčku procesu. Tímto způsobem lze snadno vzdáleně vypnout server.

Třetí možnost je nejdůležitější, kdy se použije jako parametr *MSISDN*, což je telefonní číslo, které uživatele zajímá. Toto číslo je použito pro získání požadovaných dat, které jsou po zpracování prezentovány na výstupu. Dále bude objasněna implementace tohoto řešení.

Po tom, co uživatel zadá *MSISDN* jako vstupní parametr, je zaslán dotaz s tímto parametrem jako klíčem do databáze, která obratem vrátí seznam *DIALOGUE_ID*, což jsou identifikátory dialogů, kterých se dané číslo účastnilo. V tomto stádiu je databázi zaslána dávka příkazů, kdy každý příkaz obsahuje dané *DIALOGUE_ID* a databáze vrátí všechny stavy žádaného dialogu. V cyklu jsou tímto způsobem získány všechny stavy dialogů, ve kterých telefonní číslo participovalo. Dotazy do databáze mají následující formát:

- *SMEMBERS msisdn:x*, kde x je hodnota parametru *MSISDN*.

- *LRANGE did:y*, kde *y* je *DIALOGUE_ID* získané z předchozího dotazu.

První dotaz vrátí seznam identifikátorů dialogů a druhý dotaz samotné stavy daného dialogu.

Klientský skript potřebuje rovněž znát adresu a port, kam se připojit, aby mohl pracovat se serverovou částí v případě parametru *shutdown* nebo s databázovou částí v případě parametru *MSISDN*. To je zajištěno na začátku skriptu, kde jsou uvedeny všechny tyto informace, lze je tudíž snadno modifikovat. Pro tyto účely je potřeba vyplnit čtyři položky, příklad může být:

- databaseHost = '127.0.0.1'
- databasePort = 6379
- serverHost = '127.0.0.1'
- serverPort = 5504

Tyto hodnoty jsou výchozí. V případě změny se musí modifikovat. Pro testovací účely se počítá s tím, že všechny části aplikace běží na stejném virtuálním stroji.

Poté, co jsou z databáze úspešně získány stavy jednotlivých dialogů, je potřeba je mapovat na čitelné názvy, z databáze jsou totiž navráceny kódy stavů, ne jejich jména. K tomu slouží překladový systém, který spoléhá na doplnění hodnot od uživatele skriptu. Pro znázornění, jak je mapování zapsáno, bude uvedena služba *Loan SMS*.

```
loanDialogueStateMapping = { '1' : 'LOAN_SMS_REQUEST_RECEIVED',
                             '2' : 'LOAN_SMS_WF_RESPONSE',
                             '3' : 'LOAN_SMS_RESPONSE_RECEIVED',
                             '4' : 'LOAN_SMS_TERMNATED', ... }

loanDialogueEventMapping = { '1' : 'LOAN_SMS_REQUEST',
                             '2' : 'LOAN_SMS_RESPONSE',
                             '3' : 'LOAN_SMS_SERVICE_COMPLETED',
                             '4' : 'LOAN_SMS_NO_KEY',
                             '5' : 'LOAN_SMS_REQUEST_REMINDER', ... }

loanMap = { 'DIALOGUE_TYPE'  : 14,
            'EVENT_TYPE'     : loanDialogueEventMapping,
            'DIALOGUE_STATE' : loanDialogueStateMapping }

servicesMap = { 'LOAN_SMS' : loanMap, ... }
```

Tímto způsobem je zapisováno mapování pro jednotlivé služby. Proměnná *serviceMap* obsahuje jméno služeb a k nim přiřazené asociativní pole, která obsahují definici mapování služby. Úkolem tohoto mapovacího systému je přeložit získaná tři čísla (*DIALOGUE_TYPE*, *EVENT_TYPE*, *DIALOGUE_STATE*) na skutečné názvy. O to se stará v tomto případě proměnná *loanMap*, která udržuje překladové informace. *DIALOGUE_TYPE* je přiřazeno přímo číslo, které označuje číslo služby. *EVENT_TYPE* a *DIALOGUE_STATE* mají své vlastní mapovací proměnné typu asociativní pole (*loanDialogueStateMapping* a *loanDialogueEventMapping*). Když se tedy překládají čísla získané z databáze před samotným

vypsáním, skript najde v tomto zápisu skutečné jméno hodnoty, aby mohl vypsat kromě čísla i skutečný název.

Výstup potom vypadá tak, že skript vypíše vždy hlavičku dialogu, kde jsou jen stručné informace o konkrétním dialogu a pod ní podrobnější informace. Konkrétně jsou v hlavičce obsaženy tyto informace:

- NODE - doménové jméno virtuálního stroje, který dialog obsluhoval.
- DIALOGUE_TYPE - číslo a název služby, o kterou jde.
- DATE - datum uskutečnění dialogu.
- TIME - čas uskutečnění dialogu.
- DIALOGUE_INITIATOR - telefonní číslo, které zaslalo zprávu.
- DIALOGUE_RECEIVER - telefonní číslo, které obdrželo zprávu.

Pod touto hlavičkou jsou další podrobnější informace. Zde jsou odděleny všechny stavy dialogu. Každý stav obsahuje čtyři řádky:

- EVENT_TYPE - typ události vyjádřen číslem i názvem.
- DIALOGUE_STATE - stav, ve kterém se dialog aktuálně nachází, vyjádřen číslem i názvem.
- DATE - datum uskutečnění události.
- TIME - čas uskutečnění události.

Tohle je formát výstupu pro pracovníky zákaznické podpory, kteří ve výpisu jednoduše vidí, v jakých stavech daný dialog byl a můžou výrazně rychleji najít chyby, než kdyby hledali ve velkých logovacích souborech serveru Xtend. Příklad jednoho z možných výstupů lze vidět v příloze na obrázku A.1.

Kapitola 7

Testování

Následující text se věnuje celkovému testování monitorovací aplikace, tudíž automatickému i manuálnímu testování. Tato kapitola se věnuje testování pouze rozhraní příkazové řádky, neboť grafické uživatelské rozhraní je nad rámec této práce a bude dokončeno v dalších fázích vývoje.

7.1 Automatické testování

Plně automatické testování v případě této aplikace použito nebylo, bylo zapotřebí rovněž lidské interakce. Na testování se podílel náš vedoucí pracovník zákaznické podpory a integrační inženýr. Xtend využívá vlastního testovacího rámce ve skriptovacím jazyku Tcl/Tk pro interní testování na aplikačním serveru Xtend. Pomocí tohoto frameworku byly opakovaně spouštěny rozsáhlé sady testů, přičemž běžela zároveň i monitorovací aplikace. Ta sbírala informace z logovacích souborů vygenerovaných spuštěnými testy. Následně se tester podíval na zápis jednotlivých testů, která čísla se v něm účastnila a pokoušel se využít klientské části k tomu, aby si zobrazil všechny stavy dialogů a zkontroloval jejich korektnost podle výsledku proběhlých testů.

7.2 Manuální testování

Aplikace byla testována převážně manuálně, kdy člověk musel kontrolovat výsledky, které nástroj poskytuje. Způsob byl takový, že si tester zkopíroval a modifikoval CSV soubor, který vyprodukoval aplikační server Xtend. Modifikace spočívala v manuálním přidání dialogů a jejich stavů. Stavy mohly být klidně logicky nesmyslné, v tom případě ovšem tester musí počítat s tím, že tyto stavy mu budou skriptem vráceny. Tyto zmodifikované CSV soubory se nechaly zpracovat serverovou částí, aby se zavedly do databáze. Následně tester využil skriptu, aby mu byly navráceny dialogy a stavy, které poté porovnal s tím, co vložil do CSV souboru.

Kapitola 8

Závěr

Na vytvoření monitorovacího nástroje vznikl původně požadavek z toho důvodu, aby zákaznická podpora a integrační inženýři nemuseli při řešení problémů na straně aplikačního serveru Xtend prohledávat tisíce řádků z logovacích souborů, které Xtend při běhu tvoří a neztráceli tak drahocený čas na něčem, co by šlo zautomatizovat. Požadavek byl na nástroj, který by zpracovával všechny informace ze souborů, které tito lidé musí běžně prohledávat, extrahoval by relevantní informace a ukládal do databáze.

V rámci bakalářské práce byly nejdříve prozkoumány způsoby zachytávání dat mezi uzly na síti, poté navržen a implementován monitorovací nástroj, který splňuje požadavky aplikačního serveru Xtend. Tento produkt vlastní firma Xura, Inc., vývojový tým se nazývá Xtend a hlavní vývoj probíhá v Brně. Server Xtend je nasazován komerčně, tudíž v této práci nejsou prezentovány informace o vnitřní implementaci.

Úvodní část se zaměřovala na existující techniky monitorování datových přenosů mezi aplikacemi na síti a nejběžnějšími monitorovacími nástroji. Následně bylo objasněno, co aplikační server Xtend ve skutečnosti je a jak vysokoúrovňově funguje. Poté byl monitorovací nástroj navržen. Návrh obsahuje vstupní data, prezentaci pro uživatele a testování aplikace. V následující implementační části byla podrobněji rozebrána databázová, serverová a klientská část, což ukončuje kapitola s popisem testování.

Nástroj ke své funkčnosti využívá vícero technologií. Serverová strana je psána v C++ se standardem z roku 2011. Klientská část v Pythonu verze 2.x. Nejsou to nejnovější standardy z toho důvodu, aby šel nástroj bez problému přeložit a spustit na jakémkoli virtuálním stroji, který v cloudu nasazujeme. Jako operační systém je používán Red Hat Enterprise Linux. Databázová část je v Redisu, pro nějž je potřeba knihovna hiredis k obsluze samotné databáze. Zatím je podporováno pouze rozhraní příkazové řádky.

Další rozšiřování monitorovacího nástroje již probíhá. Aplikace se aktuálně integruje s ostatními produkty firmy, aby byla zajištěna komunikace napříč všemi produkty a mohl tak tuto aplikaci obsluhovat každý. Tato integrace je nutná prerekvizita k tomu, aby mohlo být implementováno grafické uživatelské rozhraní, které bude fungovat jako webová aplikace. Ta bude mít serverovou část v jazyce Java a uživatelskou část v Javascriptovém nástroji Angular.

Literatura

[1] Darwen, H.: *An Introduction to Relational Database Theory*. bookboon.com, 2009, ISBN 978-87-403-0777-1.

[2] Donahoo, M. J.: *TCP/IP Sockets in C, Second Edition: Practical Guide for Programmers*. Morgan Kaufmann, 2009, ISBN 978-0123745408.

[3] Garcia-Molina, H.: *Database Systems: The Complete Book*. Pearson Prentice Hall, 2009, ISBN 9780131873254.

[4] Lutz, M.: *Learning Python*. O'Reilly Media, 2013, ISBN 978-1449355739.

[5] Sadalage, P. J.: *NoSQL Distilled*. Addison-Wesley, 2012, ISBN 9780133036121.

[6] Sanders, C.: *Practical Packet Analysis*. No Starch Press, 2011, ISBN 978-1593272661.

[7] Stevens, W. R.: *TCP/IP Illustrated, Volume 1*. Addison–Wesley, 1993, ISBN 978-0201633467.

[8] Stevens, W. R.: *TCP/IP Illustrated, Volume 2*. Addison–Wesley, 1995, ISBN 978-0201633542.

[9] Stevens, W. R.: *Unix Network Programming: The Sockets Networking API*. Addison–Wesley, 2003, ISBN 978-0131411555.

[10] Stroustrup, B.: *The C++ Programming Language*. Addison-Wesley, 2013, ISBN 978-0321563842.

[11] WWW stránky: *MongoDB Manual*. [Online; navštíveno 16.04.2017]. URL https://docs.mongodb.com/manual/

[12] WWW stránky: *MySQL Documentation*. [Online; navštíveno 16.04.2017]. URL https://dev.mysql.com/doc/

[13] WWW stránky: *Network tap*. [Online; navštíveno 10.03.2017]. URL https://en.wikipedia.org/wiki/Network_tap

[14] WWW stránky: *PostgreSQL Documentation*. [Online; navštíveno 16.04.2017]. URL https://www.postgresql.org/docs/

[15] WWW stránky: *Redis Documentation*. [Online; navštíveno 16.04.2017]. URL https://redis.io/documentation

[16] WWW stránky: *Relace (matematika)*. [Online; navštíveno 12.04.2017]. URL https://cs.wikipedia.org/wiki/Relace_(matematika)

[17] WWW stránky: *Tcpdump Man Page*. [Online; navštíveno 12.03.2017].
URL http://www.tcpdump.org/tcpdump_man.html

[18] WWW stránky: *Wireshark Wiki*. [Online; navštíveno 11.03.2017].
URL https://wiki.wireshark.org/CaptureSetup

Přílohy

Příloha A

Ukázka použití rozhraní příkazové řádky

```
rnd-vm-10-66# python client.py 421724000007
-----------------------------------------
NODE:                           ADM_NODE1
DIALOGUE_TYPE:                  (4) GIFT_SMS
DATE:                           2016-09-02
TIME:                           14:12:16
DIALOGUE_INITIATOR:             421724000009
DIALOGUE_RECEIVER:              421724000007

+++++

EVENT_TYPE:                     (1) GIFT_SMS_REQUEST
DIALOGUE_STATE:                 (1) GIFT_SMS_REQUEST_RECEIVED
DATE:                           2016-09-02
TIME:                           14:12:16

EVENT_TYPE:                     (2) GIFT_SMS_STARTED
DIALOGUE_STATE:                 (1) GIFT_SMS_REQUEST_RECEIVED
DATE:                           2016-09-02
TIME:                           14:12:16

EVENT_TYPE:                     (4) GIFT_SMS_RECIPIENT_PROVISION_OK
DIALOGUE_STATE:                 (2) GIFT_SMS_TERMINATED
DATE:                           2016-09-02
TIME:                           14:12:17
-----------------------------------------


-----------------------------------------
NODE:                           ADM_NODE1
DIALOGUE_TYPE:                  (1) COLLECT_SMS
DIALOGUE_TYPE:                  (4) GIFT_SMS
DATE:                           2016-09-02
TIME:                           14:46:17
DIALOGUE_INITIATOR:             421724100003
DIALOGUE_RECEIVER:              421724000007

+++++

EVENT_TYPE:                     (1) GIFT_SMS_REQUEST
DIALOGUE_STATE:                 (1) GIFT_SMS_REQUEST_RECEIVED
DATE:                           2016-09-02
TIME:                           14:46:17

EVENT_TYPE:                     (2) GIFT_SMS_STARTED
DIALOGUE_STATE:                 (1) GIFT_SMS_REQUEST_RECEIVED
DATE:                           2016-09-02
TIME:                           14:46:17

EVENT_TYPE:                     (3) GIFT_SMS_ORIGINATOR_CHARGE_ERR
DIALOGUE_STATE:                 (2) GIFT_SMS_TERMINATED
DATE:                           2016-09-02
TIME:                           14:46:17

EVENT_TYPE:                     (1) COLLECT_REQUEST
DIALOGUE_STATE:                 (1) COLLECT_REQUEST_RECEIVED
DATE:                           2016-09-02
TIME:                           14:46:17

EVENT_TYPE:                     (2) COLLECT_RESPONSE
DIALOGUE_STATE:                 (1) COLLECT_REQUEST_RECEIVED
DATE:                           2016-09-02
TIME:                           14:46:17
-----------------------------------------
```

Obrázek A.1: Vyhledání stavů dialogů spojených s číslem 421724000007

Příloha B

Návod k použití

Nejprve je potřeba zajistit všechny technologické prerekvizity:

- GCC verze minimálně 4.8.4.
- Python verze 2.x.
- Redis verze minimálně 3.x.
- Knihovna hiredis verze 0.13.x.

Sestavit serverovou část ze zdrojových souborů lze programem make příkazem *make no_oam*. Parametr *no_oam* je použit z toho důvodu, aby sestavená aplikace nevyžadovala přítomnost produktu OAM, který se stará o spojení monitorovacího nástroje s ostatními produkty. U zákazníka je toto chování ovšem vyžadováno, tudíž je doporučený postup sestavit aplikaci pouze pomocí *make*.

Redis se může nakonfigurovat podle aktuálních požadavků s perzistencí dat nebo bez. Ve většině případů není perzistence dat vyžadována, tudíž se může Redis nakonfigurovat pouze jako cache databáze, aby nebyl zbytečně vytěžován pevný disk.

Před samotným spuštěním aplikace je potřeba její konfigurace v souboru config.ini. Struktura konfiguračního souboru bude popsána v příloze C. Následně se aplikace spustí s parametrem:

--ini=path

path zde představuje cestu ke konfiguračnímu souboru, kde jsou obsaženy všechny potřebné definice. Ty jsou detailněji popsány rovněž v příloze C.

Příloha C

Konfigurační soubor

Konfigurační soubor je potřeba ke spuštění aplikace. Obsahuje určité definice chování, které si serverová část načte a podle nich následně pracuje. Zde budou uvedeny výchozí hodnoty, které se mohou změnit v závislosti na požadavcích:

- path = /sdr/monitor - cesta k adresáři, který je monitorován a kam budou kopírovány CSV soubory.
- redis_host = 127.0.0.1 - IP adresa rozhraní, kam se má proces připojovat ke spojení s Redisem.
- redis_port = 6379 - port procesu, kde běží Redis.
- redis_expire_time = 86400 - expirační doba záznamů v Redisu v sekundách.
- post_processing_delete = yes - určuje, zda mají být zpracované soubory smazány.
- pre_processing = yes - umožňuje zpracovat soubory, které již jsou v monitorovacím adresáři, když se aplikace spustí.

Pod těmito definicemi jsou uvedeny cesty ke konfiguračním souborům služeb aplikačního serveru Xtend. Jde o soubory **.ini* a **.ads*, které jsou nutné ke správné integraci serverové části s ostatními produkty jiných týmů.

Příloha D

Obsah CD

Soubory obsažené na přiloženém CD:

- src/ - zdrojové soubory monitorovacího nástroje.
- latex/ - zdrojové soubory této práce.
- xkocma04-xtend.pdf - elektronická verze této práce.